CS2013 - Programación III

Práctica Calificada #1 (PC1)

2022 - 1

Profesor: Rubén Rivas

Templates Funciones y Contenedores - 8 puntos

Desarrollar el template de función **generate_polynomial** que permita generar una función que retorne el valor de un polinomio f(x) para un valor determinado de **x.** el template de función podría tener dos o más parámetros que permitan generar el polinomio:

$$f(x) = a + bx + cx^2 + dx^3 + ex^4 + fx^5 + gx^6 + \cdots$$

donde:

$x = variable$
$a, b, c, d, e, f, \ldots = coeficientes\ de\ los\ miembros$

- El **primer parámetro** será un **parámetro variadic no tipo** (int) donde cada valor entero representará el coeficiente de cada uno de los miembros del polinomio. Así por ejemplo si se deseara generar el siguiente polinomio:

  $$f(x) = 10 + 7x^2 + 2x^3 + 11x^6$$

  Los valores de sus coeficientes serian:

  $$osea\ a = 10, b = 0, c = 7, d = 2, e = 0, f = 0, g = 11$$

- El **ultimo parámetro** será un **parámetro tipo** que generalizará el tipo de dato de **x** y será deducible del valor del parámetro de la función.

```
template <int … Coefficients, typename T>
T generate_polynomial (T x){ …
```

Parámetros de Template
• Parámetro variadic No-Tipo int (entero) en caso de recursividad podría requerirse un parámetro adicional
• Parámetro tipo

Caso de uso #1:

```
cout << generate_polynomial<10, 0, 7, 2, 0, 0, 11>(5) << endl;
```

Equivaldría a:

$f(5) = 10 \times 5^0 + 0 \times 5^1 + 7 \times 5^2 + 2 \times 5^3 + 0 \times 5^4 + 0 \times 5^5 + 11 \times 5^6 = 172310$

Caso de uso #2:

```
cout << generate_polynomial<1, 0, 2, 3>(4.0) << endl;
```

Equivaldría a: $f(4) = 1 + 2 \times 4^2 + 3 \times 4^3 = 225$

Caso de uso #3:

```
cout << generate_polynomial<1, 2, 3>(10.0f) << endl;
```

Equivaldría a: $f(10) = 1 + 2 \times 10^1 + 3 \times 10^2 = 321$

Clases y Templates - 12 puntos

Basados en la pregunta anterior, desarrollar la clase template **polinomial** que a partir de una lista de valores x genere un arreglo dinámico de pares ordenado x, f(x) (se sugiere utilizar std::pair para crear el par ordenado).

La clase contará con 2 constructores con parámetros: uno del tipo **std::initializer_list** y otro del tipo **std::vector,** donde se ingresarán la lista de valores x que permitirán generar por cada valor de x un par ordenado x, f(x). El valor de f(x) se generará utilizando el template de funciones **generate_polynomial** desarrollado en la pregunta anterior.

La clase template contará con 2 parámetros de template colocados de forma inversa a la función **generate_polynomial** debido a que el primer parámetro no podrá ser deducido:

- El **primer parámetro** será un **parámetro tipo** que generalizará el tipo de dato de **x** y será deducible del valor del parámetro de la función.
- El **segundo parámetro** será un **parámetro variadic no tipo** (int) donde cada valor entero representará el coeficiente cada uno de los miembros del polinomio. Así por ejemplo si se deseara generar el siguiente polinomio:

```
template <typename T, int … Coefficients>
class polynomial { …
```

la clase debe contar con los atributos adicionales que se requieran.

La clase deberá ser implementada utilizando **arreglos dinámicos** (NO usar ninguno de los contenedores de la librería estándar).

Los métodos que deben implementarse son los siguientes:

- **Sobrecarga del operador +=,** que permita ingresar el valor de x y agregar un par ordenado x, f(x) adicional al arreglo de pares ordenados.
- **void clear(),** que permita eliminar todos los términos del polinomio.

Además la clase debe contar con los constructores correspondientes para permitir copiar, mover, asignar un objeto e implementar el polinomio con el constructor por default.

Debe implementarse la sobrecarga del **operador << ostream** de modo que al ejecutarla debe mostrar el par ordenado en el formato $\{x_1, f_1(x)\}\{x_2, f_2(x)\}\ldots\{x_n, f_n(x)\}$ ordenados ascendentemente.

Parámetros de Template
• Parámetro tipo
• Parámetro variadic No-Tipo int (entero)

Atributos
• Arreglo dinámico de pares ordenados pair<T, T>
• Size (contador de pares ordenados)
• Otros si considera necesario

Métodos
• Constructor por default
• Constructor con parámetro std::initializer_list
• Constructor con parámetro std::vector
• Constructor copia y asignación copia
• Destructor
• Sobrecarga de operador +=
• Sobrecarga de operador <<
• Método clear()
• Otros si considera necesario

Caso de uso #1:

```
vector<int> vec = { 1, 4, 5, 2, 3};
polynomial<int, 1, 0, 3, 4> p1 = vec; // f(x) = 1 + 3x^2 + 4x^3
cout << p1 << endl; // {1, 8} {2, 45} {3, 136} {4, 305} {5, 576}
```

Caso de uso #2:

```
polynomial<int, 0, 1> p1 = {1, 2, 3, 4}; // f(x) = x
cout << p1 << endl; // {1, 1} {2, 2} {3, 3} {4, 4}
```

Caso de uso #3:

```
vector<int> vec = { 1, 4, 5, 2, 3};
polynomial<int, 1, 0, 3, 4> p1; // f(x) = 1 + 3x^2 + 4x^3
for (const auto& item: vec)
        p1 += item;
cout << p1 << endl; // {1, 8} {2, 45} {3, 136} {4, 305} {5, 576}
```

Caso de uso #4:

```
polynomial<double, 0, 1, 1, 1, 0, 1> p1 = {1.0, 2.0, 6.0, 8.0};
cout << p1 << endl; // {1, 4} {2, 46} {6, 8034} {8, 33352}
polynomial<double, 0, 1, 1, 1, 0, 1> p2 = p1; // f(x) = x+ x^2 + x^3 + x^5
p2 += 3.5;
cout << p2 << endl; // {1, 4} {2, 46} {3.5, 583.844} {6, 8034} {8, 33352}
cout << p1 << endl; // {1, 4} {2, 46} {6, 8034} {8, 33352}
```

Caso de uso #5:

```
polynomial<double, 0, 1, 1, 1, 0, 1> p1 = {1.0, 2.0, 6.0, 8.0};
cout << p1 << endl; // {1, 4} {2, 46} {6, 8034} {8, 33352}
p1.clear();
p1 += 11.0;
cout << p1 << endl; // {11, 162514}
```

Barranco, 26 de abril del 2022