Estructuras de Datos

Programación Genérica en C++
(Templates o Plantillas)

Prof. Luis Garreta

Ingeniería de Sistemas y Computación
Pontificia Universidad Javeriana – Cali

9 de octubre de 2017

Introducción

Tres paradígmas de programación anteriores, estas son:

- programación clásica o procedimental,
- programación estructurada y
- programación orientada al objeto POO.

Programación genérica

- La programación genérica está mucho más centrada en los algoritmos que en los datos, y su postulado fundamental puede sintetizarse en una palabra: **generalización**.
- Significa que, en la medida de lo posible, los algoritmos deben ser parametrizados al máximo.
- Algoritmos expresados de la forma más independiente posible de detalles concretos
- Esto permite que los algoritmos puedan servir para la mayor variedad posible de tipos y estructuras de datos.

Ejemplo Programación Clásica

```
int max(int x, int y)
{
    return (x < y) ? y : x;
}

float max(float x, float y)
{
    return (x < y) ? y : x;
}
```

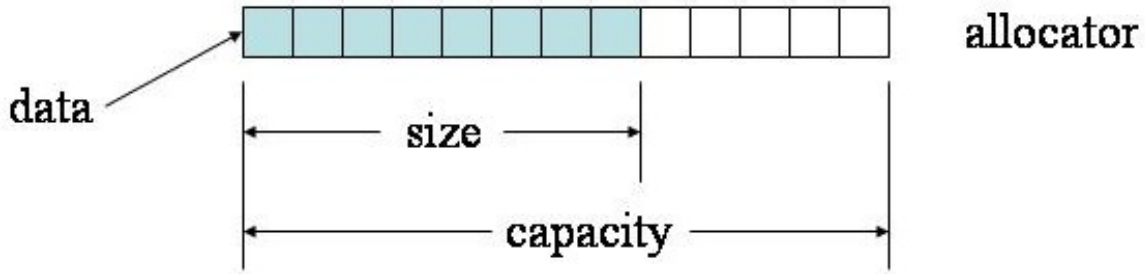
Programación Genérica (Templates C++)

- Los templates son muy importantes en el desarrollo de software con C++.
- Es sin duda una de las caracteristicas mas importantes (si no es la mas importante) de este lenguaje.
- Nos permite escribir codigo generico que puede ser usado con varios tipos de datos.
- Sin templates, se tendrian que reescribir muchas funciones y clases.
- Tomemos el siguiente ejemplo de una funcion que retorna el maximo de dos valores:

```
template <typename T>
T max(T x, T y)
{
    return (x < y) ? y : x;
}

int main () {
  float a = maximo (5.5,6.4);
  cout << a << endl;
  return 0;
}
```

Templates en Clases: Estructura de Datos Vector (Arreglo)



Funciones que sean las encargadas de la manipulación de los datos dentro del vector. Para comenzar, podemos pensar en las funciones:

- **mostrar**, para desplegar o imprimir los elementos del vector.
- **ordenar**, para ordenar los elementos del vector.
- **buscar**, para determinar la presencia o ausencia de un determinado elemento dentro del vector.
- **insertar**, para agregar componentes al vector.
- **eliminar**, para quitar componentes del vector.
- **capacidad**, para obtener la capacidad máxima del vector.
- **cuenta**, para obtener el número de elementos actuales contenidos por el vector.

Vector con Estructuras

```
typedef struct vector {
    int capacidad;
    int cuenta;
    int *data;
};

int mostrar(vector *v);

int main () {
  vector vec;
  ...
  mostrar (vec);
  ...
}
```

Vector con Objetos

```
// vector.cpp

class vector {
    int capacidad;
    int cuenta;
    int *data;
public:
  vector()  {
        capacidad = DEFAULT_SIZE;
        cuenta = 0;
        data = new int[DEFAULT_SIZE];
  }
  ~vector() { delete[] data; }
  void mostrar()  {
    for (int t = 0; t < cuenta; t++) {
      cout << "elemento " << t;
      cout << " valor " << data[t] <<
           endl;
    }
  }
  ...
}
```

```
// mainvector.cpp

#define MAX 10
int main()
{
    vector v;
    srand( time(NULL) );
    for (int r = 0; r < MAX; r++)
        v.insertar( rand() % 100);

    cout << "\nvector v sin ordenar\n";
    v.mostrar();

    v.ordenar();
    cout << "\nvector v ordenado\n";
    v.mostrar();

    getchar();
    return 0;
}
```

Una plantilla para la clase vector

```
// vector.h

#define DEFAULT_SIZE 128

template <class T> class vector {
    // atributos
    int capacidad;
    int cuenta;
    T   *data;

public:
    // constructor base
    vector()     {
      capacidad = DEFAULT_SIZE;
      cuenta = 0;
      data = new T[DEFAULT_SIZE];
    }

    // destructor
    ~vector() { delete[] data; }

    void mostrar();
    int  insertar(T d);
    void ordenar();
};
```

```
// vector.cpp

// implementación del método insertar
template <class T> int vector<T>::insertar(T d)
     {
    if (cuenta == capacidad) return -1;
    data[cuenta] = d;
    cuenta ++;
    return cuenta;
}
// implementación del método ordenar
template <class T> void vector<T>::ordenar(){
    T temp;
    int i, j, fin = cuenta;

    i = 0;
    while (i < fin )    {
        for ( j = i ; j < fin-1; j++)
        if ( data[j] > data[j+1] )        {
            temp = data[j];
            data[j] = data[j+1];
            data[j+1] = temp;
        }
        fin --;
    }
}
```

Uso de la Plantilla Vector

```
#define TEST 10
int main() {
    // prueba de un vector de números de punto flotante
    vector<double> v;
    srand( time(NULL) );
    for (int r = 0; r < TEST; r++) v.insertar( (rand() % 10) * 0.5);
    cout << "\nvector v sin ordenar\n";
    v.mostrar();
    v.ordenar();
    cout << "\nvector v ordenado\n";
    v.mostrar();

    // prueba de un vector de números long int
    vector<long int> v2;
    srand( time(NULL) );
    for (int r = 0; r < TEST; r++) v2.insertar( (rand() % 100) );
    cout << "\nvector v2 sin ordenar\n";
    v2.mostrar();
    v2.ordenar();
    cout << "\nvector v2 ordenado\n";
    v2.mostrar();

    return 0;
}
```

Tarea

Implementar el resto de las funciones de la clase Vector:

- **mostrar**, para desplegar o imprimir los elementos del vector.
- **ordenar**, para ordenar los elementos del vector.
- **buscar**, para determinar la presencia o ausencia de un determinado elemento dentro del vector.
- **insertar**, para agregar componentes al vector.
- **eliminar**, para quitar componentes del vector.
- **capacidad**, para obtener la capacidad máxima del vector.
- **cuenta**, para obtener el número de elementos actuales contenidos por el vector.