Templates

Taller de Programación I - FIUBA

Cuales son las diferencias?

```
class Array_ints {
  int data[64];
public:
  void set(int p, int v) {
    data[p] = v;
  }
  int get(int p) {
    return data[p];
  }
};
```

```
class Array_chars {
  char data[64];
public:
  void set(int p, char v) {
    data[p] = v;
  }
  char get(int p) {
    return data[p];
  }
};
```

Espacios distintos: 64***sizeof(int)** contra 64***sizeof(char)**

Llaman a código (operador asignación) distintos.

Otros también: el constructor por copia, etc

Templates

```
template<class T>
class Array {
  T data[64];
public:
  void set(int p, T v) {
    data[p] = v;
  }
  T get(int p) {
    return data[p];
  }
};
```

```
class Array_chars {
  char data[64];
public:
  void set(int p, char v) {
    data[p] = v;
  }
  char get(int p) {
    return data[p];
  }
};
```

Con sólo **Array<T>** se puede generar automáticamente variables del mismo código.

Array<int> crea un array de 64 **int**;

Array<char> crea un array de 64 **char** (como la clase Array_chars)

Optimización por tipo - Especialización de templates

```
template<>
class Array<bool> {
  bool data[64];
public:
  void set(int p, bool v) {
    if (v) { data[p/8] = data[p/8] | (1 << (p%8));  }
    else { data[p/8] = data[p/8] & ~(1 << (p%8));  }
  }
  bool get(int p) {
    return (data[p/8] & (1 << (p%8))) != 0;
  }
};
```

Array<bool> usará el código especializado.

Con cualquier otro tipo se usará el código genérico **Array<T>**

Esto permite tener código eficiente para tipos particulares.

Se requiere implementar primero **Array<T>** antes de la especialización.

Polimorfismo en tiempo de compilación

```
template<class T>
bool cmp(T a, T b) {
    return a == b;
}

template<>
bool cmp(const char* a, const char* b) {
    return strncmp(a, b, SZ);
}
```

La especialización de templates es también para tener implementaciones más apropiadas para un tipo en particular.

cmp(*“foo”*, *“foo”*) podría dar **false** con la implementación genérica por comparar sólo punteros

La especialización para **const char*** es más apropiada.

Funciona como polimorfismo pero en tiempo de compilación.

Especialización parcial de templates

```
template<class T>
class Array<T*> : private Array<void*> {
public:
  void set(int p, T* v) {
    Array<void*>::set(p, v);
  }
  T* get(int p) {
    return (T*) Array<void*>::get(p);
  }
};
```

Cuando se quiera un array de punteros, se usará el código especializado para ellos (**Array<T*>**).

Esta especialización parcial reutiliza la especialización completa para **void***. El compilador directamente no genera código lo que evita el **code bloat**.

Se requiere implementar primero **Array<T>** y **Array<void*>** antes de la especialización.

Resumen

- Jamás implementar un template al primer intento. Crear una clase prototipo (**Array_ints** , testearla y luego pasarla a template **Array<T>**
- Si se va a usar el templates con punteros, evitar el **code bloat** implementando la especialización **void*** (**Array<void*>**) y luego la especialización parcial **T*** (**Array<T*>**).
- Opcionalmente, implementar especializaciones optimizadas (**Array<bool>**)