Templates (clases)

Templates (clases)

- Al igual que en templates (funciones), el código de las clases template debe colocarse en un fichero .h (header)
- La implementación de funciones miembro “template” pueden colocarse en un fichero .hpp (no compilable) con un comando *#include* al final del fichero .h

Ejemplo

.h file

```
#ifndef SOMECLASS_H
#define SOMECLASS_H
// class definition

template <class T>
class SomeClass
{
    …….code here……
}

#include “SomeClass.hpp”
#endif
```

.hpp file

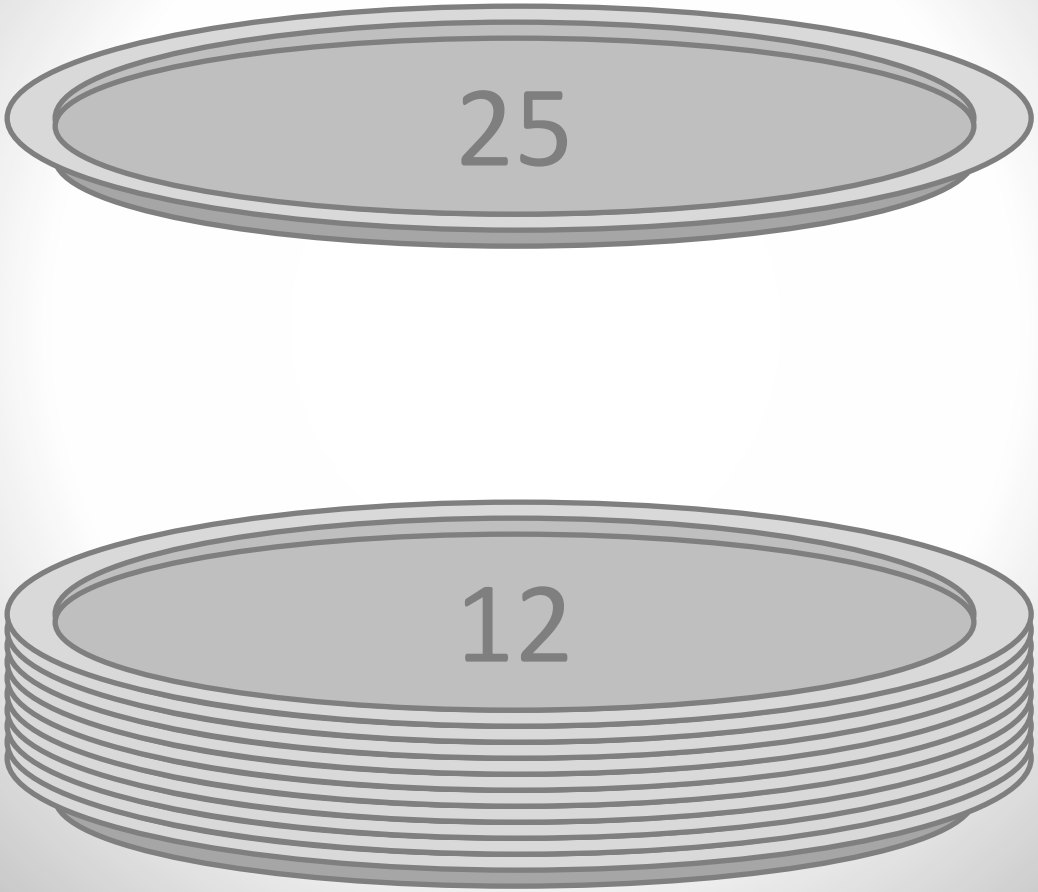
```
// definition of first function
template <class T>
void SomeClass<T>::a_function()
{
    ……. code here …..
}

// definition of second function
template <class T>
int SomeClass<T>::another_function()
{
   ……. code here …….
}
```

Sintaxis

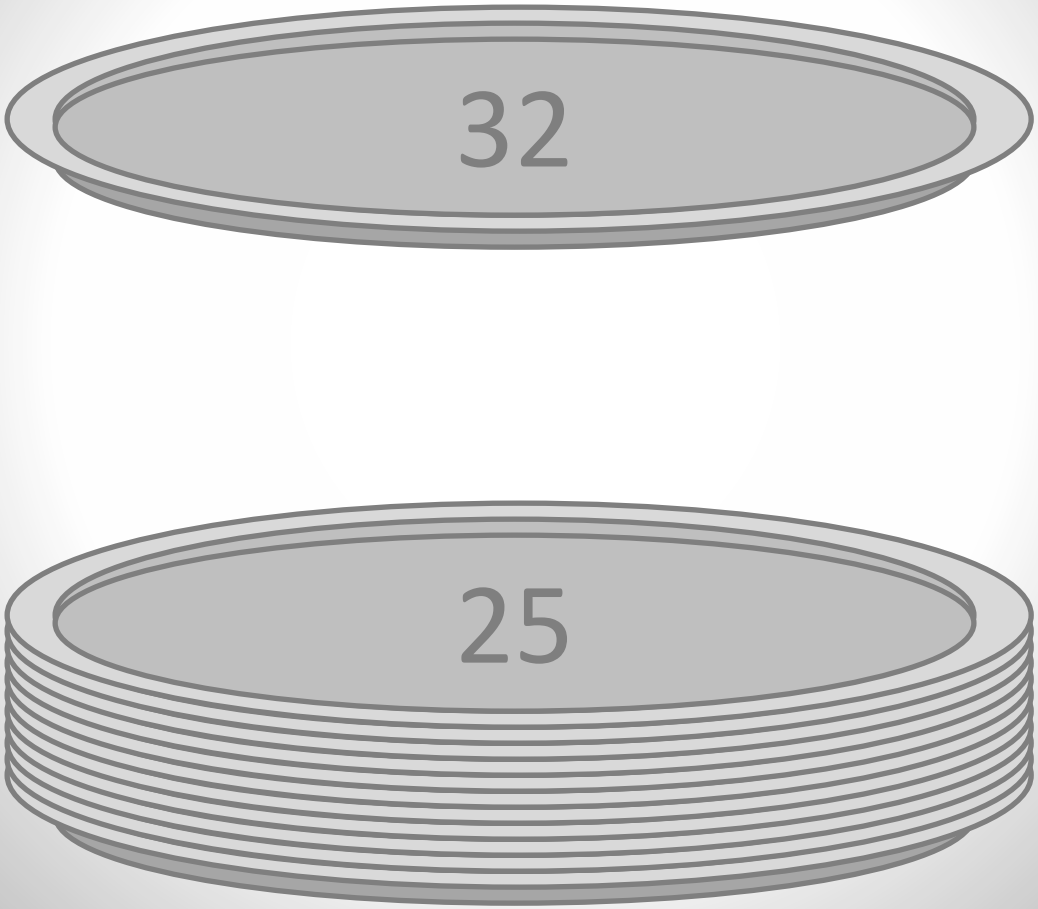
- La definición de la clase debe ser etiquetada como template
- Las funciones miembro deben ser “template” en caso necesario

Ejemplo: pila

Push



Push



Pop

Pop

Clase template “pila”

```
template <class T_element>
class stack
{
 private:
  T_element m_data[MAX];
  int m_numElements;
 public:
  stack():m_numElements(0){}
  stack(const stack<T_element>& source);
  bool push(const T_element elm);
  bool pop(T_element & elm);
  bool isFull()const {return m_numElements == MAX;}
  bool isEmpty()const {return m_numElements == 0;}
};
```

Push

```
template <class T_element>
bool stack<T_element>::push(const T_element elm)
{
    bool ret = false;
    if (!isFull())
    {
        m_data[m_numElements] = elm;
        m_numElements++;
        ret = true;
    }

    return ret;
}
```

Push

```
// stack.hpp
template <class T_element>
bool stack<T_element>::push(const T_element elm)
{
    bool ret = false;
    if (!isFull())
    {
        m_data[m_numElements] = elm;
        m_numElements++;
        ret = true;
    }

    return ret;
}
```

Pop

```
// stack.hpp
template <class T_element>
bool stack<T_element>::pop(T_element & elm)
{
    bool ret = false;
    if (!isEmpty())
    {
        elm = m_data[m_numElements-1];
        m_numElements--;
        ret = true;
    }

    return ret;
}
```

Pop

```
// stack.hpp
template <class T_element>
bool stack<T_element>::pop(T_element & elm)
{
    bool ret = false;
    if (!isEmpty())
    {
        elm = m_data[m_numElements-1];
        m_numElements--;
        ret = true;
    }

    return ret;
}
```

Copy Constructor

```
template <class T_element>
stack<T_element>::stack(const stack<T_element>& source)
{
    m_numElements = source.m_numElements;

    for (int i=0; i < m_numElements; i++)
        m_data[i] = source.m_data[i];
}
```

Copy Constructor

```
template <class T_element>
stack<T_element>::stack(const stack<T_element>& source)
{
    m_numElements = source.m_numElements;

    for (int i=0; i < m_numElements; i++)
        m_data[i] = source.m_data[i];
}
```

Copy Constructor

```
template <class T_element>
stack<T_element>::stack(const stack<T_element>& source)
{
    m_numElements = source.m_numElements;

    for (int i=0; i < m_numElements; i++)
        m_data[i] = source.m_data[i];
}
```

Copy Constructor

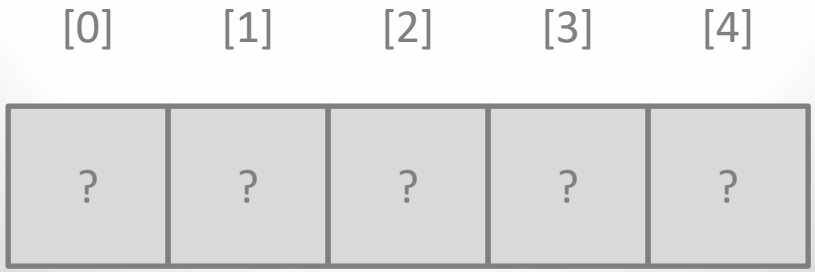
```
template <class T_element>
stack<T_element>::stack(const stack<T_element>& source)
{
    m_numElements = source.m_numElements;

    for (int i=0; i < m_numElements; i++)
        m_data[i] = source.m_data[i];
}
```

Ejemplo de uso: pila de enteros

```
stack<int> ant_hill;
int a_value;
cout << ant_hill.pop(a_value) << endl;
ant_hill.push(5);
ant_hill.push(6);
cout << ant_hill.pop(a_value) << endl;
cout << a_value << endl;
```

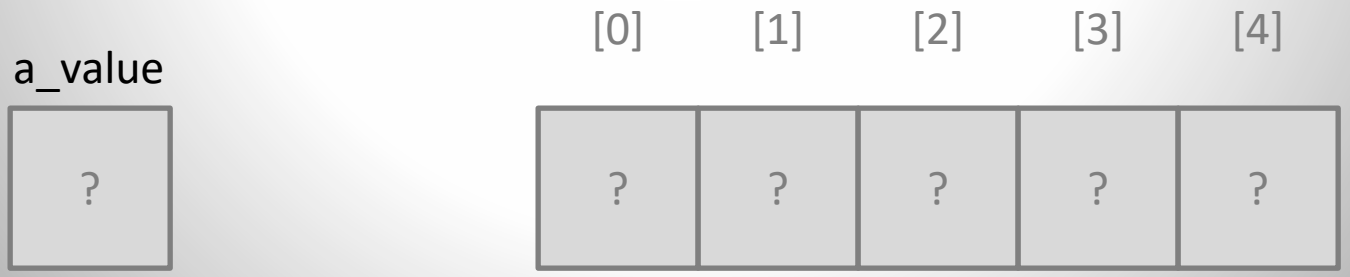
Ejemplo de uso: pila de enteros

```
stack<int> ant_hill;
int a_value;
cout << ant_hill.pop(a_value) << endl;
ant_hill.push(5);
ant_hill.push(6);
cout << ant_hill.pop(a_value) << endl;
cout << a_value << endl;
```



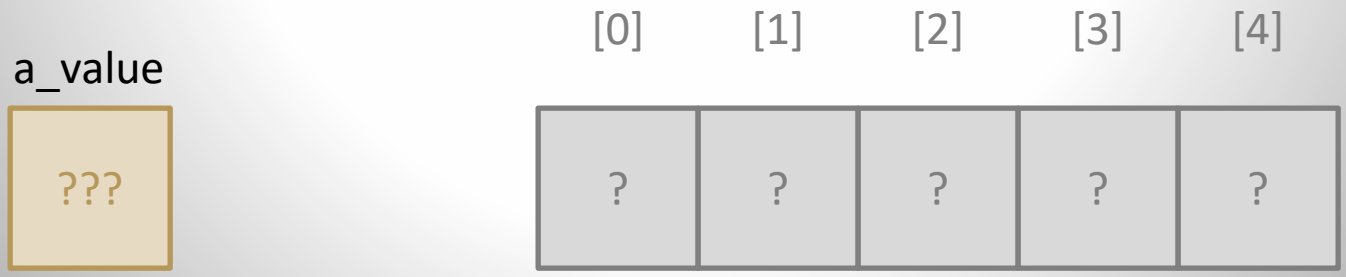
Ejemplo de uso: pila de enteros

```
stack<int> ant_hill;
int a_value;
cout << ant_hill.pop(a_value) << endl;
ant_hill.push(5);
ant_hill.push(6);
cout << ant_hill.pop(a_value) << endl;
cout << a_value << endl;
```



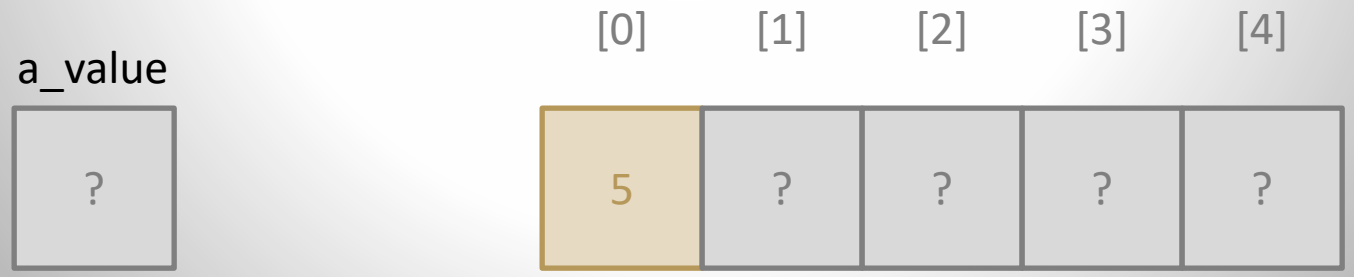
Ejemplo de uso: pila de enteros

```
stack<int> ant_hill;
int a_value;
cout << ant_hill.pop(a_value) << endl;
ant_hill.push(5);
ant_hill.push(6);
cout << ant_hill.pop(a_value) << endl;
cout << a_value << endl;
```



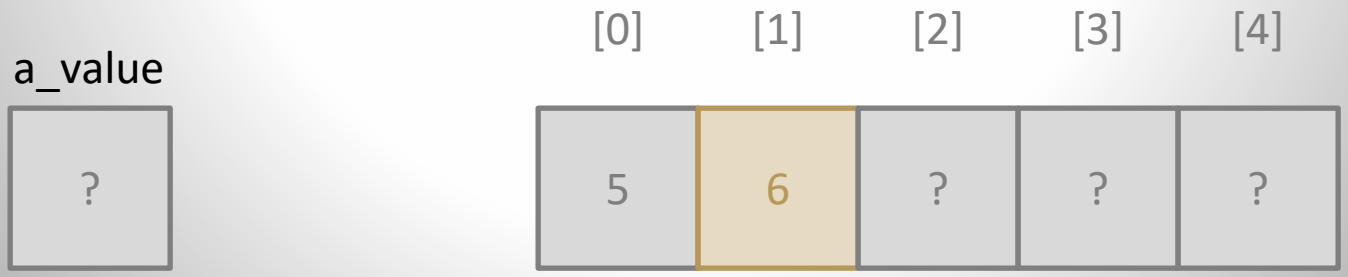
Ejemplo de uso: pila de enteros

```
stack<int> ant_hill;
int a_value;
cout << ant_hill.pop(a_value) << endl;
ant_hill.push(5);
ant_hill.push(6);
cout << ant_hill.pop(a_value) << endl;
cout << a_value << endl;
```



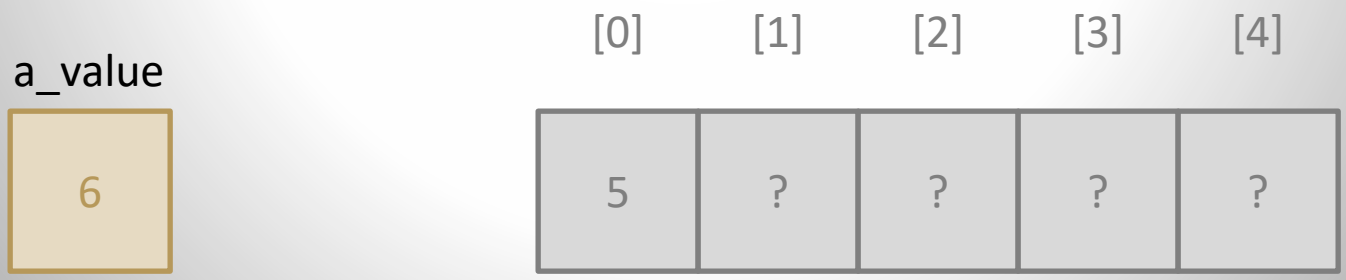
Ejemplo de uso: pila de enteros

```
stack<int> ant_hill;
int a_value;
cout << ant_hill.pop(a_value) << endl;
ant_hill.push(5);
ant_hill.push(6);
cout << ant_hill.pop(a_value) << endl;
cout << a_value << endl;
```



Ejemplo de uso: pila de enteros

```
stack<int> ant_hill;
int a_value;
cout << ant_hill.pop(a_value) << endl;
ant_hill.push(5);
ant_hill.push(6);
cout << ant_hill.pop(a_value) << endl;
cout << a_value << endl;
```



Ejemplo de uso: pila de elefantes

```
class Elephant
{
    private:
        char m_name[20];
        float m_wt;
    public:
        Elephant(): m_wt(300.5) { strcpy(m_name, "Pepe");};
        Elephant(char name[], float wt): m_wt(wt) { strcpy(m_name, name);
   };
        friend ostream& operator<<(ostream& os, const Elephant& ele);
};

ostream& operator<<(ostream& os, const Elephant& ele)
{
    os << ele.m_name << "||" << ele.m_wt;
    return os;
}
```

Ejemplo de uso: pila de elefantes

```
stack<Elephant> pack_o_derms;
Elephant ele1("Juan", 400.9);
Elephant ele2("Luis", 345.7);
pack_o_derms.push(ele1);
pack_o_derms.push(ele2);
Elephant ele3;
cout << pack_o_derms.pop(ele3) << endl;
cout << ele3 << endl;
```