## Templates (funciones)

## Sobrecarga vs. Templates

- La sobrecarga de funciones permite que varias funciones compartan el mismo nombre, teniendo distinto tipo/número de parámetros
- Las funciones **template** permiten que una sola función pueda tener parámetros de distintos tipos, conservando la misma funcionalidad

## Templates

- Las funciones template no tienen declaración
- El código de funciones template debe colocarse en un fichero “header”

## Sintaxis

```
template <typename T>
void gswap ( T& t1, T& t2)
{
        T temp = t1;
        t1 = t2;
        t2 = temp;
        return;
}
```

## Sintaxis

```
template <typename T>
void gswap ( T& t1, T& t2)
{
    T temp = t1;
    t1 = t2;
    t2 = temp;
    return;
}
```

## Ejemplo

```
template <typename T>
void gswap (T& t1, T& t2)
{
    T temp = t1;
    t1 = t2;
    t2 = temp;
    return;
}

int main()
{
    int a = 8, b = 3;
    float c = 4.3, d = 98.5;
    char c1 ='y', c2 = '@';

    gswap( a, b );
    gswap( c1, c2 );
    gswap( c, d );
    return 0;
}
```

## Ejemplo

```
// T=int
void gswap (int& t1, int& t2)
{
    int temp = t1;
    t1 = t2;
    t2 = temp;
    return;
}

int main()
{
    int a = 8, b = 3;
    float c = 4.3, d = 98.5;
    char c1 ='y', c2 = '@';

    gswap( a, b );
    gswap( c1, c2 );
    gswap( c, d );
    return 0;
}
```

## Ejemplo

```
// T=char
void gswap (char& t1, char& t2)
{
    char temp = t1;
    t1 = t2;
    t2 = temp;
    return;
}

int main()
{
    int a = 8, b = 3;
    float c = 4.3, d = 98.5;
    char c1 ='y', c2 = '@';

    gswap( a, b );
    gswap( c1, c2 );
    gswap( c, d );
    return 0;
}
```

## Ejemplo

```
// T=float
void gswap (float& t1, float& t2)
{
    float temp = t1;
    t1 = t2;
    t2 = temp;
    return;
}

int main()
{
    int a = 8, b = 3;
    float c = 4.3, d = 98.5;
    char c1 ='y', c2 = '@';

    gswap( a, b );
    gswap( c1, c2 );
⇨   gswap( c, d );
    return 0;
}
```

## Definición de operaciones

```
// Pre: The type to fill the template parameter must have
//      the “insertion” operator defined for it.
template <typename U>
void print (const U & u)
{
    cout << u;
    return;
}

// Pre: The type of the template parameter must have the
//      “<“ operator defined.
template <typename T_type>
T_type min_value (const T_type & t1, const T_type & t2)
{
    return (t1 < t2 ? t1 : t2);
}
```

## Templates de varios tipos

```
// Pre: Both template types must have insertion
//      operator defined.
template <typename T, typename U>
void repeater (const int num_times, const T t1, const U u1)
{
    for (short i = num_times; i > 0; i--)
        cout << t1 << " " << u1 << endl;
    return;
}
```

```
// Pre: Both template types must have insertion
//      operator defined.
template <typename T, typename U>
void repeater (const int num_times, const T t1, const U u1)
{
    for (short i = num_times; i > 0; i--)
        cout << t1 << " " << u1 << endl;
    return;
}

int main()
{
    char some_character = '$';
    float a_float_value = 2.2;
    repeater(4, some_character, a_float_value);

    return 0;
}
```

```
// Pre: Both template types must have insertion
//      operator defined.
template <typename T, typename U>
void repeater (const int num_times, const T t1, const U u1)
{
    for (short i = num_times; i > 0; i--)
        cout << t1 << " " << u1 << endl;
    return;
}

int main()
{
    char some_character = '$';
    float a_float_value = 2.2;
    repeater(4, some_character, a_float_value);

    return 0;
}
```

output

```
$ 2.2
$ 2.2
$ 2.2
$ 2.2
```