# Unidad 1

# Fundamentos de Programación Competitiva

# Logro de sesión

Al finalizar la sesión, el estudiante comprenderá conceptos de Standard Template Library

## Semana 3

# Standard Template Library

**Contenido**:

- Template
- Contenedores
- Iteradores

# Standard Template Library

- Colección de estructuras de datos y algoritmos escritos en C++.
- Estos contenedores se dice que son genéricos porque pueden contener instancias de cualquier otro tipo de dato.
- Las aplicaciones pueden escribirse rápidamente ya que se construyen a partir de algoritmos eficientes y los programadores pueden seleccionar el algoritmo más rápido para una situación dada.

# Template

Código que puede ser utilizado con diferentes tipos de datos.

```
#include "iostream"
#include "conio.h"
using namespace std;
template<typename T>
T suma(T a, T b) {
        return a + b;
}
int main() {
        double x = 10.5, y = 20.75;
        cout << "\nEntero: " << suma<int>(x, y)<<"\n";
        cout << "\nFloat: " << suma<float>(x, y) << "\n";
        cout << "\nDouble: " << suma<double>(x, y) << "\n";

        return 0;
}
```

# Template

```
#include "iostream"
#include "conio.h"

using namespace std;
//Usando Clases como tipo de dato
class persona{
   public:
        string nombres;
        bool estado;
        persona(){
                this->nombres="";
                this->estado=false;
        }
        ~persona(){}
};

//Definimos el template
   template <typename T>
   T activar(T objeto){
        objeto.estado=true;

        return objeto;
   }
```

```
int main(){
   //Creamos la instancia con el estado falso
   persona* objPersona=new persona();
        objPersona->nombres="Carlos";
        objPersona->estado=false;

   //Imprimimos el valor actual de estado
   cout << "Valor actual de estado: " <<
objPersona->estado << endl;

   //Usamos el método template
   *objPersona=activar<persona>(*objPersona);

   //Imprimimos el valor del nuevo estado
   cout << "Nuevo valor de estado: " <<
objPersona->estado;

   return 0;
}
```

# Contenedores

- Estructura que puede almacenar una colección de elementos del mismo tipo. Se dividen en 3 categorías:

  - **Contenedores Lineales:** Almacenan los objetos de forma secuencial.
  - **Contenedores Asociativos:** Tiene asociada una clave, mediante la clave se pueden almacenar al contendedor.
  - **Contenedores Adaptados:** Cambiar un contenedor en un nuevo contenedor modificando la interface.

# Iteradores

- Es un puntero generalizado que identifica una posición en un contenedor.
- Nos permite recorrer un contenedor en la dirección que se nos permita

Contenedor.begin() → Iterador al primer elemento

Contenedor.end() → Iterador al final del contenedor

Iterador++ → Ir al siguiente elemento

Iterador- - → Ir al elemente anterior

*iterador → Acceder al valor de un elemento

# Pair

- Agrupar dos elementos.
- Un pair es un término que se utiliza para combinar dos valores que pueden o no ser del mismo tipo.

```
int main()
{
	pair<int, char> PAIR1;
	PAIR1.first = 100;
	PAIR1.second = 'G';
	cout << PAIR1.first << " ";
	cout << PAIR1.second << endl;
	return 0;
}
```

# Vector

- Está implementando con como un bloque de memoria contiguo de forma similar a un array.
  - Inserciones y borrados en tiempo constante al principio y al final.
  - Inserciones y borrados de coste lineal en posiciones intermedias.
  - Gestión automática de memoria.

# Vector

```
vector<int> punts_equipos;          //declarar vector
int numEquipos;
cin >> numEquipos;
while(numEquipos > 0)
{ int puntos;
 cin >> puntos;
 punts_equipos.push_back(puntos); //agregar un elemento al final
 numEquipos = numEquipos - 1;
 punts_equipos.pop_back(); //eliminar ultimo elemento
 punts_equipos.front();   //obtener primer elemento
 punts_equipos.back();   // obtener el ultimo elemento
 punts_equipos.erase(punts_equipos.begin() + pos);  //eliminar
 elemento en posición pos
}
```

# Set

- Permite almacenar varios valores bajo una sola variable, pero cada valor sólo se almacena una vez.

- Los elementos son guardados en orden ascendente

```
set<string> s_nombres;  //declarar set
string nombre;
int i = 0;
while (i < 5)
{ cin >> nombre;
s_nombres.insert(nombre);  //insertar elemento
s.nombres.erase(nombre);   //eliminar elemento
i = i + 1;
}
cout << "Numero de elementos en set: " <<
s_nombres.size() << endl;  //tamaño set
```

# Map

- Conocido como Diccionario
- Permite almacenar pares de forma (llave, valor), donde la clave es única.

```
map<string, string> paises_moneda;
paises_moneda["Espana"] = "euro";
paises_moneda["EEUU"] = "dolar";
paises_moneda["Singapur"] = "dolar";
paises_moneda["Inglaterra"] = "libra";
paises_moneda["Egipto"] = "libra";
cout << "La moneda usada en Egipto es: " <<
paises_moneda["Egipto"] << endl;
```

# Muchas Gracias!!!