```
/* Tema 1 Arrays
    Arrays de bajo nivel
        - Especificando tamaño char v[5]
        - Sin especificar tamaño char v[]
    El tamaño puede ser una variable statica y contante, ejemplo:
        static const int TAM=50;
        private int vectorPrivado[TAM];
*/
    //Paso de arrays : NO ES POR VALOR; similar a por referencia.

    //Funcion:
    int obtenerValores(int valores[]){
        for(int i=0;i<totalUtilizados;i++)
            valores[i]=vectorPrivado[i];
        return totalUtilizados;
    }
    //Main
    main(){
        int datosObjeto[TAM];
        int d=objeto.obtenerValores(datosObjeto);
    }

/*-------------------------------------------------------------------------------------------*/
/* Tema 2 Punteros
1.- Inicializacion
    ptr=&x;
    prt=prt2;
    ptr=0;
2.- Solo se puede hacer la asignacion si hay coincidencia de tipos
        double x;
        int *ptr;
        ptr=&x; =>ERROR
3.- Unico valor directo que se les puede asignar es 0 o NULL;

4.- Relacion entre punteros y arrays, ejemplo:
    * un nombred e un array se puede ver comoun puntero constante, al que no se le puede
    asignar una direccion de memoria diferente.

    char x;
    char array[5]={1,2,3,4,5};
    *array="b"; //array[0]="b", actuan de la misma manera, ya que el nombre del array actua
    como un puntero constante al inicio del array.

NOTA: Podemos acceder a las siguientes posiciones o modificar donde apunta el puntero:
    *(array+1) <- Los punteros se almacenan contiguamente en memoria.
    *(array++) <- Apunta a la siguiente direccion de memoria sin modifica el puntero (SOLO
    LECTURA Y ESCRITURA; NO CAMBIA EL PUNTERO)
*/
    //Ejemplo de uso:
        const int tam=10;
        int numero[tam];
        for(int i=0;i<tam;i++)
            cin>>*(numeros+1);


    // Usar otro puntero para hacer lo mismo:
    int *ptr;
    ptr=numeros; //apunta a numeros[0];
    for(int i=0;i<tam;i++)
        cin>>*(ptr+1);
    //Podemos cambiar donde apunta el puntero:
    ptr=numeros+1; //Sin * ni parentesis por que ambos actuan como punteros
    ptr=&numeros[1]; // Como accedemos a una posicion en la que se nos devuelve un valor,
    debemos usar & para acceder a la direccion de memoria


    //EJEMPLOS USO PUNTEROS;
    int main(){
        int a=5;*p=&a;**PP=&p;
        **pp=*p+(**pp/a);   //-> a=5+(5/5) = 6
        *p=a+1;             //a=6+1;
        a=**pp/2;           //a=7/2 = 3;
    }
```

```
    int main(){
        int a=5,*p=&a;
        *p=*p*a; //-> a=a*a=25
    }


/* 5.- Aritmetica de punteros
Las operaciones que permiten los punteros son:
    * suma (puntero+1)  - resta(puntero-1)
    * "++"
    * &array[1]>&array[0] TRUE      array<&array[4] TRUE    array==&array[0] TRUE
*/
    //Ejemplo de uso:
    const int Tam=10;
    int conjunto[Tam]={1,52,48,2,456,9,489,1,584,18};
    int *ptrInt;
    ptrInt=conjunto; //aputna al 0
    for(int i=0;i<Tam);i++){            while(ptrInt<&conjunto[Tam]){
        cout << *ptrInt;                    cout << *ptrInt;
        ptrInt++;                           ptrInt++;
    }                                   }
    for(int i=0;i<Tam);i++){            while(ptrInt>conjunto){
        ptrInt--;                           ptrInt--;
        cout << *ptrInt;                    cout << *ptrInt;
    }                                   }


/*
    Ejemplo funciones con punteros:
    DECLARACION:
        void obtenerValorPuntero(int *);
    IMPLEMENTACION:
    void obtenerValorPuntero(int * valor){
        cout << "Meter valor";
        cin >> *valor;
    }
    int main(){
        int numero;
        obtenerValorPuntero(&numero); //Hay que especificar el & ya que pide un puntero;
    }
*/

/*
8.- Punteros a punteros
int *ptr;
int **ptrptr;
*(*ptrptr)=20;

9.- Gestion dinamica de memoria, La memoria se pude gestionar reservando de forma dinamica con
el operador "new"
*/
    //Ejemplo:
    int tam;
    cout zz "Gime tamaño";
    cin >> tam;
    int *ptr = new int[tam];
/*
Liberacion se hace con el operador "delete":
    -Normal delete(ptr)
    -Array delete[]ptr;

Los errores frecuentes son:
        - Intento de acceso a memoria libreada
        - No liberacion de memoria
*/
/*-------------------------------------------------------------------------------------------*/


/* MATRICES CON PUNTEROS Y ARRAYS */
/*  Ejemplo 1.- Unico array y un puntero a el*/

class Matriz{
```

```
    private:
    int nfilas,ncolumnas;
    int *m;
    public:
    ~Matriz();
    Matriz(int,int);
    int AccederElemento(int,int);
};

//Constructor
Matriz::Matriz(int nfilas,int ncolumnas){
    this->nfilas=nfilas;
    this->ncolumnas=ncolumnas;
    m=new int[nfilas*ncolumnas];
}
//Destructor
Matriz::~Matriz(){
    this->nfilas=0;
    this->ncolumnas=0;
    delete []m;
}
//Acceso
int Matriz::AccederElemento(int fila, int columna){
    return m[fila*nfilas+columna];
}

/*  Ejemplo 2.- Array de punteros que apuntan a un array con todos los datos*/

class Matriz{
    private:
    int nfilas,ncolumnas;
    int **m;
    public:
    ~Matriz();
    Matriz(int,int);
    int AccederElemento(int,int);
};


//Constructor
Matriz::Matriz(int nfilas,int ncolumnas){
    this->nfilas=nfilas;
    this->ncolumnas=ncolumnas;
    m=new *int[nfilas];
    m[0]=new int[nfilas*ncolumnas];
    for(int i=1;i<nfilas;i++){
        m[i]=m[i-1]+ncolumnas;
    }
}
//Destructor
Matriz::~Matriz(){
    this->nfilas=0;
    this->ncolumnas=0;
    delete []m[0];
    delete []m;
}
//Acceso
int Matriz::AccederElemento(int fila, int columna){
    return m[fila][columna];
}



/*  Ejemplo 3.- Array de punteros que apuntan a las filas AMPLIADO*/

class Matriz2D{
    private:
    int nfilas,ncolumnas;
    int **matriz;
    public:
    ~Matriz2D();
    Matriz2D(int,int);
    int AccederElemento(int,int);
```

```
    bool AsignarValor(int,int,int);
    void MostrarPantalla();
    Matriz2D * copiarMatriz();
    Matriz2D * ExtraerSumbatriz(int,fIni,int fFin,int cIni,int cFin);
    void EliminaFila(int fila);
};


//Constructor
Matriz2D::Matriz2D(int nfilas,int ncolumnas){
    this->nfilas=nfilas;
    this->ncolumnas=ncolumnas;
    matriz=new *int[nfilas];
    for(int i=0;i<nfilas;i++){
        matriz[i]=new int[ncolumnas];
    }
}
//Destructor
Matriz2D::~Matriz2D(){
    this->nfilas=0;
    this->ncolumnas=0;
    for(int i=0;i<nfilas;i++)
        delete []matriz[i];
    delete []matriz;
}
//Acceso
int Matriz2D::AccederElemento(int fila, int columna){
    return matriz[fila][columna];
}

//Assignar valor
int Matriz2D::AsignarValor(int fila, int columna,int valor){
    bool salida=false;
    if(fila<nfilas&&columna<ncolumnas){
        matriz[fila][columna]=valor;
        salida=true;
    }
    return salida;
}

//Mostrar por pantalla
void Matriz2D::MostrarPantalla(){
    cout << endl;
    for(int i=0;i<nfilas;i++)
        for(int j=0;j<ncolumnas;j++)
            cout << matriz[i][j]<<" ";
        cout << endl;

}
// Copiar matriz
Matriz2D * Matriz2D::copiarMatriz(){
    Matriz2D * salida=new Matriz2D(nfilas,ncolumnas);
    for(int i=0;i<nfilas;i++)
        for(int j=0;j<ncolumnas;j++)
            resultado->matriz[i][j]=matriz[i][j];
    return salida;
}

// ExtraerSumbatriz
Matriz2D * Matriz2D::ExtraerSumbatriz(int,fIni,int fFin,int cIni,int cFin);
    Matriz2D * salida;


    if(fIni>=0&&fFin>=0&&cIni>=0&&cFin>0=&&fIni<nfilas&&fFin><nfilas&&cIni>ncolumnas&&cFin>ncolu
    mnas&&)
        salida=new Matriz2D(fIni-fFin+1,cIni-cFin+1);
    for(int i=fIni;i<fFin;i++)
        for(int j=cIni;j<cFin;j++)
            resultado->matriz[i][j]=matriz[i][j];
    return salida;
}

// Eliminar fila;
```

```
void Matriz2D::EliminaFila(int fila){
    int ** matrizNueva = new int*[nfilas-1];
    //reserva de espacio para las filas
    for(int i=0;i<nfilas-1;i++){
        matrizNueva[i]=new int[ncolumnas];
    }
    int filaNueva;
    for(int i=0;i<nfilas;i++){
        for(int j=0;j<ncolumnas;j++){
            if(i<fila)  filaNueva=i;
            else        filaNueva=i-1;
            matrizNueva[filaNueva][j]=matriz[i][j];
        }
    }
    //Liberamos espacio de matriz antigua
    for(int i=0;i<nfilas;i++)
        delete []matriz[i];
    delete[]matriz;
    matriz=matrizNueva;
    nfilas--;
}
```