

# **UNIDAD II (1)**

**METODOLOGÍA PARA SOLUCIÓN DE  
ALGORITMOS, TIPOS DE DATOS E  
IDENTIFICADORES**

# UNIDAD II

## Conceptos de Programación:

- Metodología para la Resolución de Problemas Algorítmicos.
- Tipos de Datos:
  - Simples.
  - Estructurados.
- Identificadores.

# UNIDAD II

## Conceptos de Programación:

- Operadores:
  - Aritméticos.
  - Relaciones.
  - Lógicos.
  - De Asignación.
- Expresiones.

# UNIDAD II

## Conceptos de Programación:

- Estructuras de Control:
  - Selectivas.
  - Repetitivas.
- Subprogramas.

# UNIDAD II

Conceptos de Programación:

- Datos Estructurados:
  - Cadenas.
  - Vectores.
  - Matrices.

# **METODOLOGÍA PARA LA RESOLUCIÓN DE PROBLEMAS ALGORÍTMICOS**

08/12/2015

De acuerdo con (Joyanes y Zahonero, 2010) estas son las fases de resolución de un problema:

### **Análisis del Problema.**

Analizar el problema teniendo presente la especificación de los requisitos dados por el cliente.

# ANÁLISIS

## Ejemplo:

Para un automóvil comprado en 20,000 dólares en 2005 se desea obtener una tabla con las depreciaciones acumuladas y los valores reales de cada año, durante los seis años siguientes suponiendo un valor de recuperación o rescate de 2, 000 dólares.

Realizar un análisis del problema, conociendo la fórmula de la depreciación anual constante  $D$  para cada año útil.

$$D = \frac{\text{costo} - \text{valor de recuperación}}{\text{vida útil}}$$

¿Qué entradas se requieren?

¿Cuál es la salida deseada?

¿Qué método produce la salida deseada?

Requisitos o requerimientos adicionales y restricciones a las solución.



# ANÁLISIS

¿Qué entradas se requieren?

Costo original.

Vida útil.

Valor de recuperación.

Año.

¿Cuál es la salida deseada?

Depreciación acumulada en cada año.

Depreciación anual por año.

Valor del automóvil en cada año.

# ANÁLISIS

¿Qué método produce la salida deseada?

#	Año	Depreciación	Depreciación Acumulada	Valor Anual
1	2006	3, 000	3, 000	17, 000
2	2007	3, 000	6, 000	14, 000
3	2008	3, 000	9, 000	11, 000
4	2009	3, 000	12, 000	8, 000
5	2010	3, 000	15, 000	5, 000
6	2011	3, 000	18, 000	2, 000

# ANÁLISIS

Requisitos o requerimientos adicionales y restricciones a las solución.

1. Puede ser que mi solución sea para cierto cantidad de dinero.
2. No se limito la depreciación anual (3, 000) ya que, esta puede ser menor de 3, 000.
3. Es exclusivamente para representar depreciación de un automóvil en un periodo dado.
4. Es sumamente importante tener los datos de entrada por lo tanto, de no tenerlos no realizará los cálculos.

## **Diseño del Algoritmo.**

Se diseña la solución que conducirá a un algoritmo que resuelva el problema.

# DISEÑO

Si en la Etapa de Análisis se determina **qué** hace el programa. En esta Etapa se determina **cómo** hace el programa las tareas solicitadas.

Comúnmente se usa el ya conocido *divide y vencerás*, técnicamente conocido como **top-down** (diseño descendente) **o modular**.

El proceso de romper el problema en cada etapa se denomina **refinamiento sucesivo**.

Un programa bien diseñado consta de:

Un programa principal (Modulo del nivel más alto encargado de llamar a submodulos).

Subprogramas.

Este diseño orienta al diseñador hacia la **Programación Modular**.

# DISEÑO

Los Módulos pueden ser Planeados, Codificados, Comprobados y Depurados de forma independiente y con diferentes programadores.

El proceso implica:

1. Programar un módulo.
2. Comprobar el módulo.
3. Si es necesario, depurar el módulo.
4. Combinar el módulo con los módulos anteriores.

Bien, entonces por ultimo el proceso que convierte los resultados del análisis en un diseño modular con refinamientos sucesivos que permitan una posterior traducción a un lenguaje se denomina ***diseño del algoritmo.***

# DISEÑO

Recordemos una pregunta importante de la etapa de Análisis:

**¿Qué método produce la salida deseada?**

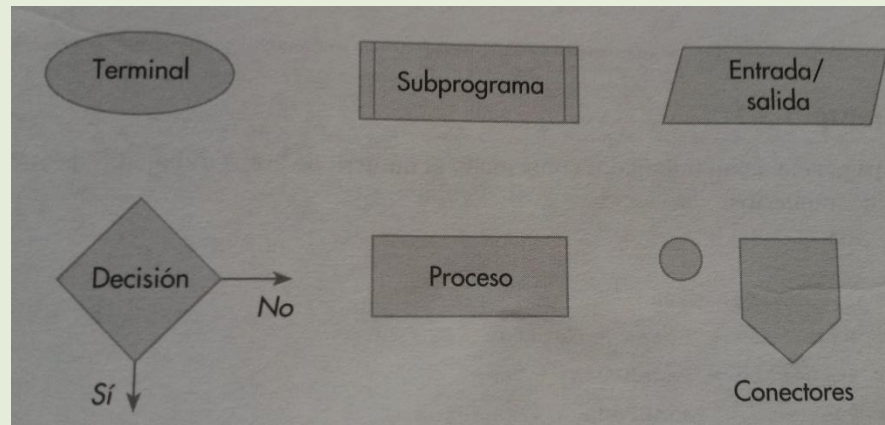
#	Año	Depreciación	Depreciación Acumulada	Valor Anual
1	2006	3, 000	3, 000	17, 000
2	2007	3, 000	6, 000	14, 000
3	2008	3, 000	9, 000	11, 000
4	2009	3, 000	12, 000	8, 000
5	2010	3, 000	15, 000	5, 000
6	2011	3, 000	18, 000	2, 000

# DISEÑO

Herramientas para diseñar:

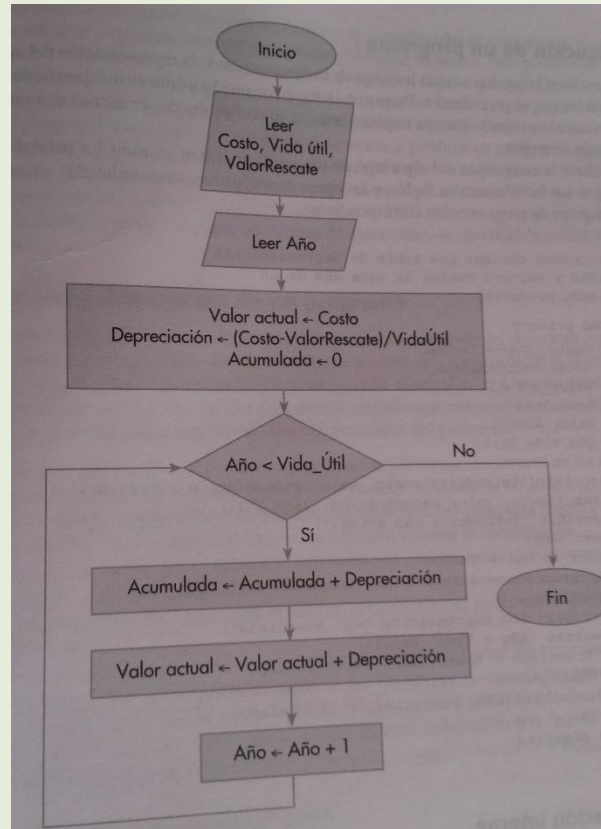
- *Diagramas de flujo.*
- *Pseudocódigos (seudocódigos).*

Se usarán Diagramas de flujo (ANSI), puesto que permite ver el algoritmo de una manera gráfica. Por lo tanto, se usarán los siguientes símbolos:

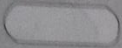
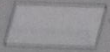
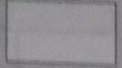
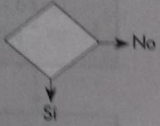
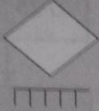








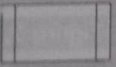

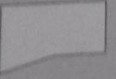
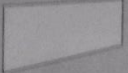
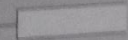
# DISEÑO



# DISEÑO

Símbolos principales	Función
	Terminal (representa el comienzo, "inicio", y el final, "fin" de un programa. Puede representar también una parada o interrupción programada que sea necesario realizar en un programa).
	Entrada/Salida (cualquier tipo de introducción de datos en la memoria desde los periféricos, "entrada", o registro de la información procesada en un periférico, "salida").
	Proceso (cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transferencia, etcétera).
	Decisión (indica operaciones lógicas o de comparación entre datos, normalmente dos, y en función del resultado de la misma determina cuál de los distintos caminos alternativos del programa se debe seguir; normalmente tiene dos salidas, respuestas SÍ o NO, pero puede tener tres o más, según los casos).
	Decisión múltiple (en función del resultado de la comparación, se seguirá uno de los diferentes caminos de acuerdo con dicho resultado).
	Conector (sirve para enlazar dos partes cualesquiera de un ordinograma a través de un conector en la salida y otro conector en la entrada). Se refiere a la conexión en la misma página del diagrama.

# DISEÑO

	Indicador de dirección o línea de flujo (indica el sentido de ejecución de las operaciones).
	Línea conectora (sirve de unión entre dos símbolos).
	Conector (conexión entre dos puntos del organigrama situado en páginas diferentes).
	Llamada a subrutina o a un proceso predeterminado (una subrutina es un módulo independiente del programa principal, que recibe una entrada procedente de dicho programa, realiza una tarea determinada y regresa, al terminar, al programa principal).
	Pantalla (se utiliza en ocasiones en lugar del símbolo de E/S).
	Impresora (se utiliza en ocasiones en lugar del símbolo de E/S).
	Teclado (se utiliza en ocasiones en lugar del símbolo de E/S).
	Comentarios (se utiliza para añadir comentarios clasificadores a otros símbolos del diagrama de flujo. Se pueden dibujar a cualquier lado del símbolo).

## **Codificación (Implementación).**

La solución se escribe en la sintaxis del lenguaje de alto nivel y se obtiene un programa fuente que se compila a continuación.

# CODIFICACIÓN

Escribir en un lenguaje de programación de la representación del algoritmo en este caso el *diagrama de flujo*.

Documentación.

Existen:

Interna.

Se incluye mediante comentarios. Deben ser *significativos*.

Externa.

Ingeniería de software.

# CODIFICACIÓN

The screenshot shows a Sublime Text editor window titled "ejemplo\_1.cpp - Sublime Text (UNREGISTERED)". The code is written in C++ and includes comments in Spanish. The code defines three functions: leerDatos, calcularDatos, and limpiarPantalla. The main function calls these functions and reads user input using scanf.

```
#include <iostream>
using namespace std;

/*Area de declaración de funciones*/
void leerDatos(float *, float *, int *, int *); /*No es necesario poner los nombres basta con los tipos*/
void calcularDatos(float, float, int, int);
void limpiarPantalla();

int main() //Inicia en mi diagrama de flujo (Punto de entrada)
{
    /*Comienza con las lecturas (Entradas)*/
    float m_fCosto, m_fValorRescate; //Almacenan $
    int m_iVidaUtil, m_iAnio; //Solo calcula años completos.

    /*Lee los datos de entrada del usuario*/
    leerDatos(&m_fCosto, &m_fValorRescate, &m_iVidaUtil, &m_iAnio); /*Se mandan las referencias*/

    limpiarPantalla();

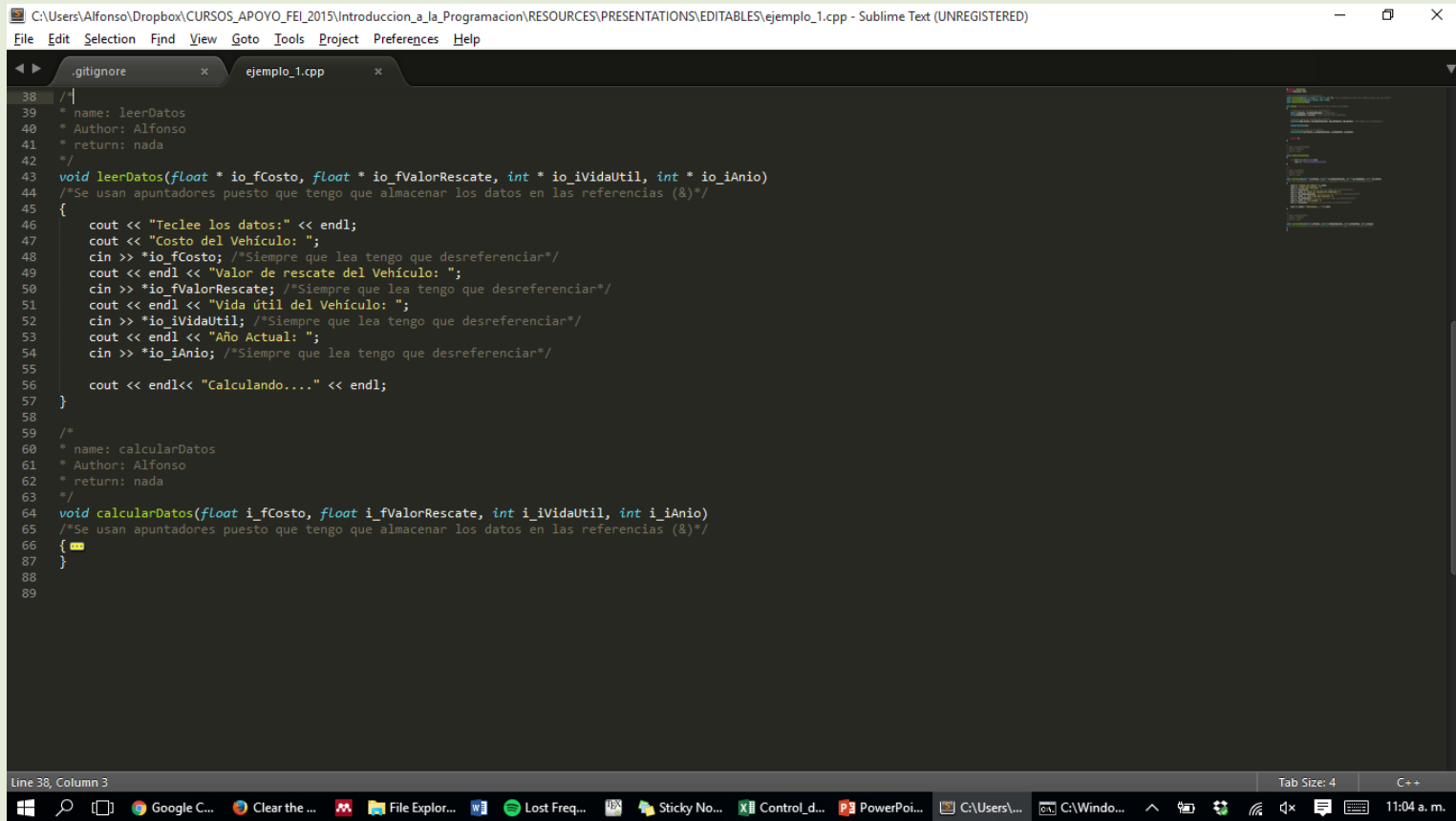
    /*Calcula los datos de la tabla*/
    calcularDatos(m_fCosto, m_fValorRescate, m_iVidaUtil, m_iAnio);

    return 0;
}

/*
 * name: limpiarPantalla
 * Author: Alfonso
 * return: nada
 */
void limpiarPantalla()
{
    for (int n = 0; n < 2; n++)
        cout << "\n\n\n\n\n\n\n\n\n\n\n";
}

/*
 * name: leerDatos
 * Author: Alfonso
 * return: nada
 */
void leerDatos(float * io_fCosto, float * io_fValorRescate, int * io_iVidaUtil, int * io_iAnio)
```

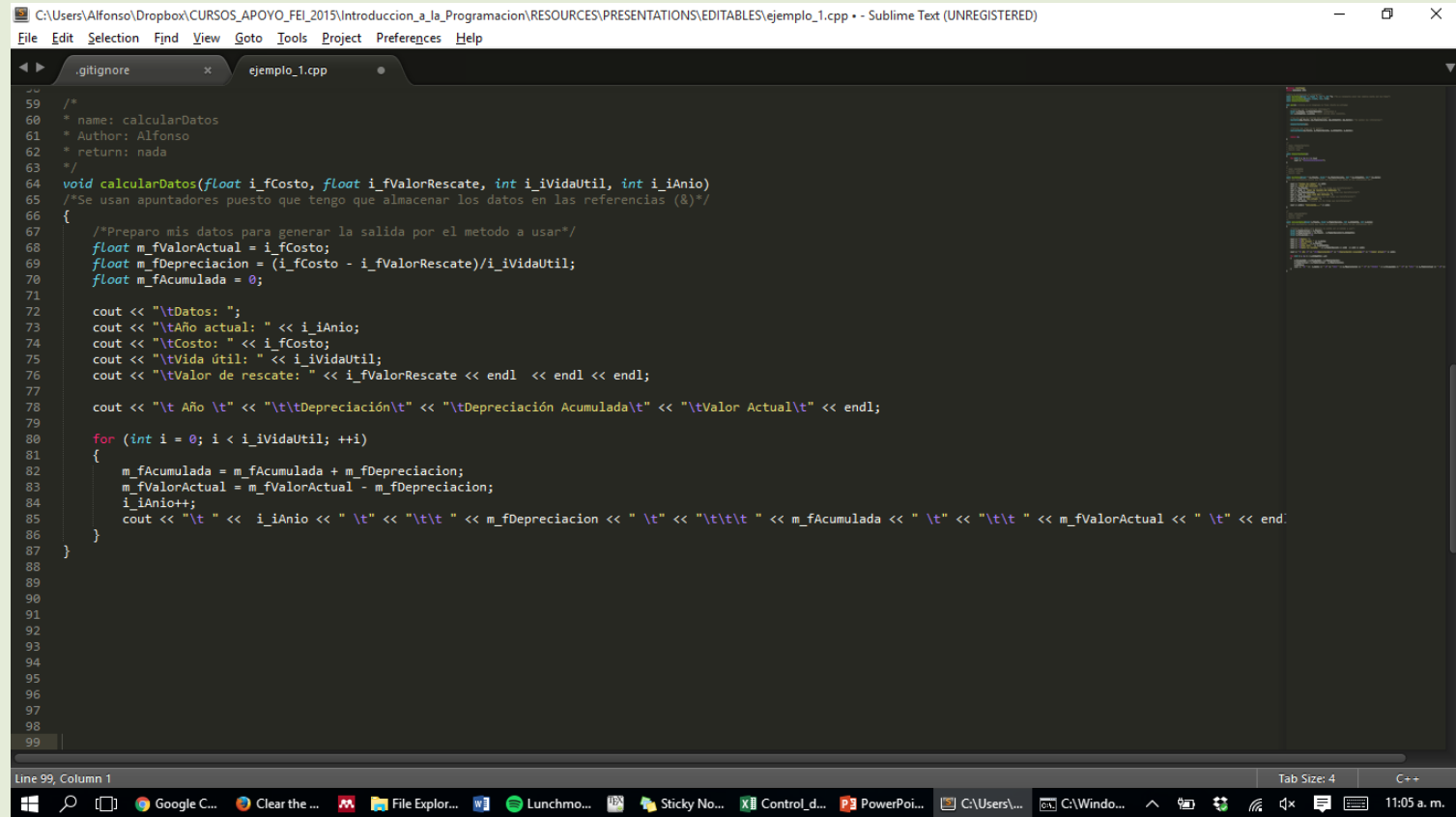
# CODIFICACIÓN



```
38  /*
39  * name: leerDatos
40  * Author: Alfonso
41  * return: nada
42  */
43  void leerDatos(float * io_fCosto, float * io_fValorRescate, int * io_iVidaUtil, int * io_iAnio)
44  /*Se usan apuntadores puesto que tengo que almacenar los datos en las referencias (&)*/
45  {
46      cout << "Teclee los datos:" << endl;
47      cout << "Costo del Vehículo: ";
48      cin >> *io_fCosto; /*Siempre que lea tengo que desreferenciar*/
49      cout << endl << "Valor de rescate del Vehículo: ";
50      cin >> *io_fValorRescate; /*Siempre que lea tengo que desreferenciar*/
51      cout << endl << "Vida útil del Vehículo: ";
52      cin >> *io_iVidaUtil; /*Siempre que lea tengo que desreferenciar*/
53      cout << endl << "Año Actual: ";
54      cin >> *io_iAnio; /*Siempre que lea tengo que desreferenciar*/
55
56      cout << endl << "Calculando...." << endl;
57  }
58
59  /*
60  * name: calcularDatos
61  * Author: Alfonso
62  * return: nada
63  */
64  void calcularDatos(float i_fCosto, float i_fValorRescate, int i_iVidaUtil, int i_iAnio)
65  /*Se usan apuntadores puesto que tengo que almacenar los datos en las referencias (&)*/
66  {
67  }
68
69  }
```

08/12/2015

# CODIFICACIÓN



```
59 /*
60 * name: calcularDatos
61 * Author: Alfonso
62 * return: nada
63 */
64 void calcularDatos(float i_fCosto, float i_fValorRescate, int i_iVidaUtil, int i_iAnio)
65 /*Se usan apuntadores puesto que tengo que almacenar los datos en las referencias (&)*/
66 {
67     /*Preparo mis datos para generar la salida por el metodo a usar*/
68     float m_fValorActual = i_fCosto;
69     float m_fDepreciacion = (i_fCosto - i_fValorRescate)/i_iVidaUtil;
70     float m_fAcumulada = 0;
71
72     cout << "\tDatos: ";
73     cout << "\tAño actual: " << i_iAnio;
74     cout << "\tCosto: " << i_fCosto;
75     cout << "\tVida útil: " << i_iVidaUtil;
76     cout << "\tValor de rescate: " << i_fValorRescate << endl << endl << endl;
77
78     cout << "\t Año \t" << "\t\tDepreciación\t" << "\tDepreciación Acumulada\t" << "\tValor Actual\t" << endl;
79
80     for (int i = 0; i < i_iVidaUtil; ++i)
81     {
82         m_fAcumulada = m_fAcumulada + m_fDepreciacion;
83         m_fValorActual = m_fValorActual - m_fDepreciacion;
84         i_iAnio++;
85         cout << "\t " << i_iAnio << " \t" << "\t\t " << m_fDepreciacion << " \t" << "\t\t\t " << m_fAcumulada << " \t" << "\t\t " << m_fValorActual << " \t" << endl;
86     }
87 }
88
89
90
91
92
93
94
95
96
97
98
99
```



## **Ejecución, Verificación y Depuración.**

Se comprueba el programa ejecutable rigurosamente y se eliminan todos los errores.

## **Mantenimiento.**

El programa se actualiza y modifica, cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de los usuarios.

## **Documentación.**

Escritura de las diferentes fases del ciclo de vida del software, esencialmente el análisis, diseño y codificación, unidos a manuales de usuario y de referencia, así como normas para el mantenimiento.

# TIPOS DE DATOS

# CONCEPTO

De acuerdo con Joyanes et al “[...] un tipo de dato es un conjunto de valores y operaciones asociadas a esos valores” (2008:2).

# PRIMITIVOS (SIMPLES)

Los tipos de datos mas simples son los *datos primitivos (simples)* también denominados *atómicos*.

*Datos atómicos:* conjunto de datos atómicos con propiedades idénticas. Se definen por un conjunto de *valores* y un conjunto de *operaciones* que actúan sobre los valores.

# PRIMITIVOS (SIMPLES)

*Diferentes tipos de datos atómicos*

## Enteros

valores	$-\infty$ , ..., -3, -2, -1, 0, 1, 2, 3, ..., $+\infty$
operaciones	*, +, -, /, %, ++, --, ...

## Coma flotante

valores	- , ..., 0.0, ...
operaciones	*, +, -, %, /, ...

## Carácter

valores	\0, ..., 'A', 'B', ..., 'a', 'b', ...
operaciones	<, >, ...

## Lógico

valores	verdadero, falso
operaciones	and, or, not, ...

# COMPUESTOS (ESTRUCTURADOS)

*Datos compuestos:* tipo opuesto a los tipos de datos atómicos. Se pueden romper en subcampos que tengan significado.

Ejemplos:

511922110101 (numero de teléfono)

(52, México) (Código de país)

# TIPOS DE DATOS (C++)

TIPO	TAMAÑO(bytes)	VALORES
bool	1	Verdadero (True) y Falso (False)
short	2	-32, 768 hasta 32, 767
int (16 bits)	2	-32, 768 hasta 32, 767
int (32 bits)	4	-2, 147, 483, 648 hasta 2, 147, 483, 647
long	4	0 hasta 4, 294, 967, 295
float	4	-1.2e-38 hasta 3.4e38
double	8	-2.2e-308 hasta 1.8e308
char	1	256 valores de carácter, si tienen signo, entonces de -128 hasta 127, si no, entonces de 0 hasta 255.



# EJEMPLO

```
.gitignore x ejemplo_1.cpp datos.cxx x
1 #include <iostream>
2
3 using namespace std;
4
5
6 int main ()
7 {
8     cout << "El tamaño de un entero es: " << sizeof(int) << endl;
9     cout << "El tamaño de un short es: " << sizeof(short) << endl;
10    cout << "El tamaño de un long es: " << sizeof(long) << endl;
11    cout << "El tamaño de un char es: " << sizeof(char) << endl;
12    cout << "El tamaño de un float es: " << sizeof(float) << endl;
13    cout << "El tamaño de un double es: " << sizeof(double) << endl;
14    cout << "El tamaño de un booleano es: " << sizeof(bool) << endl;
15
16    int arreglo[10];
17    cout << "El tamaño de un entero es: " << sizeof(arreglo)/sizeof(int) << endl;
18
19    return 0;
20 }
21
```

Line 1, Column 1

# COMPUESTOS (ESTRUCTURADOS)

Los *tipos agregados* son tipos cuyos valores constan de colecciones de elementos de datos. Existen tres tipos de datos agregados básicos: *arrays* (arreglos), secuencias (cadenas) y registros.

*Array o arreglo*: colección de datos de tamaño fijo o longitud fija, cada uno de sus datos es accesible en tiempo de ejecución.

[0, 85, 63, 78, 20, 45, 78, 20, 30]

# COMPUESTOS (ESTRUCTURADOS)

*Secuencias o cadenas:* en esencia es un array cuyo tamaño puede variar en tiempo de ejecución.

"Hola mundo!"

*Registro:* puede contener elementos agregados y primitivos. Un registro se puede considerar como un tipo o colección de datos de tamaño fijo. Sus campos pueden ser de diferentes tipos de datos. Es el tipo de dato mas próximo a la idea de objeto.

```
Registro {  
    Dato1  
    Dato2  
}
```

# IDENTIFICADORES

08/12/2015

# IDENTIFICADORES

De acuerdo con Joyanes et al “[...] es una secuencia de caracteres, letras, dígitos y subrayados (\_). El primer carácter debe ser una letra”.

Las letras minúsculas y mayúsculas son diferentes:

nombre\_clase

elemento\_mayor

a

indice

C es sensible a mayusculas. Por lo tanto C reconoce como diferentes a ALFA, alfa y ALFa.

# UNIDAD II

## Conceptos de Programación:

- Metodología para la Resolución de Problemas Algorítmicos.
- Tipos de Datos:
  - Simples.
  - Estructurados.
- Identificadores.

# REFERENCIAS

Joyanes Aguilar, Luis y Zahonero Martínez, Ignacio (2010). Programación en C, C++, Java y UML. México, McGraw-Hill/INTERAMERICANA EDITORES, S.A. DE C.V.

Joyanes Aguilar, Luis y Zahonero Martínez, Ignacio (2008). Estructuras de datos en Java. México, McGraw-Hill/INTERAMERICANA DE ESPAÑA, S. A. U.

# **UNIDAD II (1)**

**METODOLOGÍA PARA SOLUCIÓN DE  
ALGORITMOS, TIPOS DE DATOS E  
IDENTIFICADORES**