



Programación Avanzada



Pontificia Universidad
JAVERIANA
Colombia

Información de la materia Programación Avanzada

Descripción de la materia	<p>En esta asignatura se enseña el análisis, diseño e implementación de algoritmos usando estructuras estáticas y dinámicas, teniendo como base previa los métodos básicos de la programación para resolver problemas. Se desarrollan algoritmos con manejo eficiente de la persistencia de la información y se aplican los conceptos fundamentales de programación orientada a objetos.</p> <p>Al finalizar el curso el estudiante estará en capacidad de:</p> <ol style="list-style-type: none">1. Presentar estrategias en la solución de problemas usando el computador como herramienta para diseñar algoritmos que utilicen estructuras de datos básicas.2. Brindar los conceptos y herramientas para solución de problemas, que utilicen una metodología de diseño orientado a objetos.3. Reconocer estrategias y habilidades clave para desarrollar de manera autónoma algoritmos
---------------------------	---



Información de la materia Programación Avanzada

Intensidad Semanal	4 horas
Trabajo independiente	5 horas
Créditos	3
Prerrequisito	Introducción a la Programación

- Evaluación de la Habilidad Conceptual y Analítica
 - Parciales
 - Parcial I 25%
 - Parcial II 25%
 - Laboratorios – Talleres 20% (1er corte: 10%, 2do corte: 10%)
- Evaluación de la Habilidad Práctica y Trabajo en Grupo
 - Proyecto Proyecto I 15%
 - Proyecto II 15%



Bibliografía recomendada

1. Luís Joyanes Aguilar, Ignacio Zahonero Martínez, Algoritmos y Estructuras de Datos: Una perspectiva en C, McGraw Hill, 2004.
2. Jorge A. Villalobos S. Diseño y Manejo de Estructuras de Datos en C. McGraw Hill. 1996.
3. Mark Allen Weiss, Data Structures and Algorithms Analysis in C++, Addison- Wesley
4. H.M. Deitel / P.J. Deitel. C++ How to Program. Cuarta Edición. Prentice Hall. 2003.
5. H.M. Deitel / P.J. Deitel, C/C++ y Java: Cómo Programar, Prentice Hall, 2004.
6. David Vandevoorde, Nicolai M. Josuttis, C++ Templates: The Complete Guide, Addison Wesley, 2003
7. Eckel, Bruce. Piensa en JAVA, cuarta edición, Prentice Hall, 2007 Beecher Karl, Computational Thinking, BCS, 2018

Textos Complementarios

1. An introduction to object-oriented programming with JavaWu, C. Thomas. 2006.
2. Introduction to JAVA programming comprehensive version. Liang, Y. Daniel. 2007.
3. Java in two semesters.Charatan, Quentin. 2006
4. Java generics and collections. Naftalin, Maurice. 2007.
5. Java the complete reference, J2SE 5 edition. Schildt, Herbert.2005.
6. Java an introduction to problem solving & programming.2005.
7. Savitch, Walter. Resolución de Problemas con C++, Segunda Edición. Prentice Hall, 2000
8. Larman, Craig, UML y Patrones, Segunda Edición, Pearson Educación, 2002.
9. Thomas Wu, Introduccion a la Programacion Orientada a Objetos con Java, McGraw Hill, 2001.



A tener en cuenta

Reglas de las entregas:

1. No se permite código resuelto en internet ni por IA
2. No está permitido copiar código de otros grupos
3. Los programas que no estén correctamente indentados tendrán -1 punto
4. El plazo de entrega no es negociable después de confirmado
5. Las entregas son por Campus, no se permite por correo
6. Se deben entregar los archivos fuente .cpp o .java según el caso y con la estructura de directorios correcta para su ejecución. Programa que no compile no se califica y se asume como no entregado
7. Al inicio de cada programa colocar los datos de los integrantes del grupo y , su respectivo número y la fecha a la cual corresponde la entrega. Si se omite la firma, tendrán -1 punto

EJ: /*

Integrantes Pedro Perez, Mariana Gomez

Grupo: 3

Fecha: yyyy/mm/dd

Taller: N

*/

8. Enviar un pantallazo de la ejecución del programa dentro de archivo Word ([ver ejemplo](#)). Si no lo envían, se resta un punto.
9. Los talleres entregados tienen nota máxima sobre 4.0, para alcanzar el 5.0 deben entregar de forma individual los programas escritos a mano (escanear y subir con la entrega).



Conceptos de Estructuras

Las estructuras son colecciones de variables relacionadas bajo un nombre. Las estructuras en C++ nos permiten almacenar información que puede ser del mismo tipo de dato Ej: **Arreglos** y **matrices** o contener diferentes tipos de datos como los **struct**

Arreglos en C++

Un arreglo es un grupo de posiciones de memoria relacionadas entre si por el hecho de que tienen el mismo nombre y almacenan el mismo tipo de datos. Adicionalmente los elementos de un arreglo se almacenan en posiciones contiguas de memoria. Su tamaño es constante en tiempo de ejecución

Declaración de un arreglo

tipoDato nombreArreglo [tamaño];

Ej: `int miArreglo[10];`

Otras formas de como declarar e inicializar arreglos

`string vector[5] = {"5", "hola", "2.7", "8,9", "adios"};`

`int vector2[] = {1,2,3,4,10,9,80,70,19};`

Ojo: Esto genera error

`int vector2[3];`

`vector2[3] = {1,5,10};`

`float vector3[5] = {10.5}; /*Esta declaración, llena con valor la posición [0] y las demás posiciones se inicializan a cero por defecto*/`



Conceptos de Estructuras

Obtener /Asignar valor de un arreglo en una posición dada

Los arreglos en C++, comienzan con el índice 0 (cero), es decir un arreglo de cinco posiciones, tiene índices del 0 al 4

Ej:

```
float incrementos[5] = {10.5, 5.1, 8.9, 10, 95.2}; //Array con 5 elementos
```

```
float numero5 = incrementos[5]; //Para acceder al elemento 5, se usa el índice 4
```

```
float primerNumero = incrementos[0]; //
```

Para cambiar el valor de un arreglo, se utiliza el numero de índice:

Ej: incrementos[3]=10; //Aquí se modifica la 4ª posición del arreglo

Recorrer un arreglo o vector del cual no conocemos la longitud.

```
#include <iostream>
using namespace std;
int main()
{ string estudiantes[] = {"pedro","juan","miguel","sara"};
  int limite =(sizeof(estudiantes)/sizeof(estudiantes[0]));
```

```
for (int i = 0; i < limite; i++)
{
    cout<<estudiantes[i]<<endl;
}
}
```



Conceptos de Estructuras

<u>Función</u>	<u>Sintaxis</u>	<u>Significado</u>
strcat	<code>char *strcat(char *destino, const char *origen);</code>	Agrega una copia de la cadena origen al final de la cadena destino.
strchr	<code>char *strchr(char *cad, char c);</code>	Busca la primera ocurrencia del ccaracter contenido en la variable "c".
strcmp	<code>int strcmp(const char *cad1, const char *cad2);</code>	Compara uno a uno los caracteres de ambas cadenas hasta completarlos todos o hasta encontrar una diferencia. Devuelve cero (0) si ambas cadenas son iguales, un valor negativo si la primera cadena es menor que las segunda y un valor positivo si la primera es mayor que la segunda.
strcpy	<code>char *strcpy(char *destino, const char *origen);</code>	Copia todos los caracteres de la cadena origen en la cadena destino, incluso el caracter de terminación.
strlen	<code>int strlen(const char *cad);</code>	Devuelve el número de caracteres de la cadena "cad". No cuenta el caracteres de terminación.
strlwr	<code>char *strlwr(char *cad);</code>	Convierte las letras contenidas en la cadena "cad" en minúsculas.
strrchr	<code>char *strrchr(char *cad, char c);</code>	Busca la última ocurrencia del caracter contenido en la variable "c".
strrev	<code>char *strrev(char *cad);</code>	Invierte los caracteres contenidos en la cadena "cad".
strstr	<code>char *strstr(const char *cad1, const char *cad2);</code>	Busca la primera ocurrencia de la cadena "cad2" en la cadena "cad1".
strupr	<code>char *strupr(char *cad);</code>	Convierte las letras contenidas en la cadena "cad" en mayúsculas.

Conceptos de Estructuras

<u>Función</u>	<u>Sintaxis</u>	<u>Significado</u>
atoi	<code>int atoi (const char * str);</code>	Convierte una cadena en entero
atol	<code>long atol (const char * str);</code>	Convierte una cadena en entero largo
atof	<code>float atol (const char * str);</code>	Convierte una cadena en double
sprintf	<code>sprintf (char * str, const char * format, data);</code>	Genera una cadena con el mismo texto que se imprimiría si se usara formato en printf, pero en lugar de imprimirse por pantalla, el contenido se almacena como una cadena en el búfer señalado por str. Ej: <code>sprintf(cadena, "%d", numero);</code>



manejoArreglosCadenas.cpp



Pontificia Universidad
JAVERIANA
Colombia

Conceptos de Estructuras

Declaración de Arreglos multidimensionales en C++

Los arreglos multidimensionales son aquellos que tienen más de una dimensión por ende la ubicación de un elemento necesita dos índices.

Estructura de una matriz de dos dimensiones

	0	1	2	3	...	n
0						
1						
2						
3						
4						
...						
m						

Declaración de un arreglo multidimensional

`tipoDato nombreArreglo [f][c];`

Ej: `int tabla [2][3] //` donde gráficamente **2** seria m (según la gráfica) y **3** seria 2

Otras ejemplos de como declarar e inicializar arreglos

```
int tabla [2][3] = {40,41,42,43,44,45};
```

```
int tabla [2][3] = { {40,41,42},{43,44,45}};
```

Obtener /Asignar valor de un arreglo multidimensional en una posición dada

Los arreglos multidimensionales en en C++, comienzan con el índice 0,0,...,n (cero)



Conceptos de Estructuras

```
int tabla2 [2][3] = { {50,51,52},{53,54,55}};  
for (int n=0;n<2;n++){  
    for (int m=0;m<3;m++){  
        cout << " ["<<n<<","<< m <<"]:"<<tabla2[n][m];  
    }  
    cout << endl;  
}
```



Conceptos de Estructuras

Estructuras en C++

Las estructuras (o registros) son tipos de datos existentes en C++ que se utilizan para manejar información que no es homogénea. Ejemplo: Para almacenar la información de un cliente no podemos utilizar un arreglo porque los tipos de datos son diferentes:

- Tipo Doc: string
- Numero Doc: String
- Nombres: String
- Sueldo: Float
- Edad: Int

Para este tipo de problema utilizamos las estructuras el cual es un tipo de dato personalizado que contiene diferentes elementos llamados *miembros*. Cada miembro de una estructura puede tener un tipo de dato diferente a los demás miembros, e incluso puede ser de otra estructura.

Uso del typedef

El operador **typedef** se utiliza regularmente para nombramiento de tipos de datos struct, pero también se puede utilizar para crear sinónimos de tipos de datos

typedef **tipo_dato** nuevoTipoDato.

Ej:

```
typedef int entero;  
entero k;
```



Conceptos de Estructuras

Declaración de una estructura

Las estructuras como tal nos sirven para crear variables basadas en ellas por ende casi siempre la creación de una estructura viene seguida de la declaración de una o mas variables de la misma.

```
struct <nombre_estructura>
{
    Tipo_dato_1 variable1;
    Tipo_dato_2 variable2;
    .....
    Tipo_dato_3 variable3;
}
```



Conceptos de Estructuras

Ejemplos de declaración de estructuras

```
struct sCliente1{  
    string tipoDoc;  
    string numeroDoc;  
    string nombres;  
    string apellidos;  
    float sueldo;  
    int edad;  
    char fecha [8];  
};  
struct sCliente1 vCliente1;
```

```
typedef struct sCliente2 {  
    string tipoDoc;  
    string numeroDoc;  
    string nombres;  
    string apellidos;  
    float sueldo;  
    int edad;  
    char fecha [8];  
};  
sCliente2 vCliente2;
```

```
struct sCliente3 {  
    string tipoDoc;  
    string numeroDoc;  
    string nombres;  
    string apellidos;  
    float sueldo;  
    int edad;  
    char fecha [8];  
}vCliente3 ;
```



Conceptos de Estructuras

Acceso a los miembros de una estructura.

El siguiente ejemplo nos muestra como acceder a los miembros de un **struct**

```
#include <string>
#include <iostream>
using namespace std;
int main(){
    struct sCliente1{
        string tipoDoc;
        string numeroDoc;
        string nombres;
        string apellidos;
        float sueldo;
        int edad;
        char fecha [8];
    };
    struct sCliente1 vCliente1;
    vCliente1.tipoDoc="cc";
    vCliente1.numeroDoc="45697";
    cout <<vCliente1.tipoDoc<<" "<< vCliente1.numeroDoc <<endl;
```

```
typedef struct sCliente2 {
    string tipoDoc;
    string numeroDoc;
    string nombres;
    string apellidos;
    float sueldo;
    int edad;
    char fecha [8];
};

sCliente2 vCliente2;
vCliente2.tipoDoc="ti";
vCliente2.numeroDoc="89741";
cout <<vCliente2.tipoDoc<<" "<< vCliente2.numeroDoc <<endl;
```



Conceptos de Estructuras

Acceso a los miembros de una estructura.

El siguiente ejemplo nos muestra como acceder a los miembros de un **struct**

```
struct sCliente3 {  
    string tipoDoc;  
    string numeroDoc;  
    string nombres;  
    string apellidos;  
    float sueldo;  
    int edad;  
    char fecha [8];  
}  
vCliente3 ;  
vCliente3={"CE","43432234","pedro","perez",2000,33,NULL};  
cout <<vCliente3.tipoDoc<<" "<< vCliente3.numeroDoc <<endl;  
}
```



Tipos de Datos

Tipos de Datos

Luis, J. A., & Aguilar, L. J. (2008). *Fundamentos de programación* (cuarta edición). McGraw-Hill Education

Los tipos de datos básicos incorporados a C son *enteros*, *reales* y *carácter*.

Tabla D.1. Tipos de datos enteros

Tipo de dato	Tamaño en bytes	Tamaño en bits	Valor mínimo	Valor máximo
signed char	1	8	-128	127
unsigned char	1	8	0	255
signed short	2	16	-32.768	32.767
unsigned short	2	16	0	65.535
signed int	2	16	-32.768	32.767
unsigned int	2	16	0	65.535
signed long	4	32	-2.147.483.648	2.147.483.647
unsigned long	4	32	0	4.294.967.295

El tipo `char` se utiliza para representar caracteres o valores integrales. Las constantes de tipo `char` pueden ser caracteres encerrados entre comillas ('A', 'b', 'p'). Caracteres no imprimibles (tabulación, avance de página, etc.) se pueden representar con secuencias de escape ('\t', '\n').

Tipo de dato	Tamaño en bytes	Tamaño en bits	Valor mínimo	Valor máximo
float	4	32	$3.4E - 38$	$3.4E + 38$
double	8	64	$1.7E - 308$	$1.7E + 308$
long double	10	80	$3.4E - 4932$	$3.4E + 4932$



Tiposdatos.cpp



BinarioADecimal.cpp



DecimalABinario.cpp

Los datos float tienen 4 bytes, formados por un bit de signo, un exponente binario de 8 bits y una mantisa de 23 bits.



Tipos de Datos

Mantisa	Exponente	Notación científica	Valor en punto fijo
1.5	4	$1.5 \cdot 10^4$	15000
-2.001	2	$-2.001 \cdot 10^2$	-200.1
5	-3	$5 \cdot 10^{-3}$	0.005
6.667	-11	6.667e-11	0.0000000000667



numeroAChar.cpp



Pontificia Universidad
JAVERIANA
Colombia

Tipos de Datos

Conversión entre tipos de datos

(Bonet Esteban, E. V. (s. f.). *Lenguaje C*. Universidad de Valencia. <https://informatica.uv.es/estguia/ATD/apuntes/laboratorio/Lenguaje-C.pdf>)

Conversión automática de tipos de datos.

El lenguaje C++ permite que en una misma expresión aparezcan diferentes tipos

de datos, encargándose el compilador de realizar las operaciones de forma correcta. Cuando en una misma expresión aparecen dos o más tipos de datos, el compilador convierte todos los operandos al tipo del operando más grande existente de acuerdo con las dos reglas siguientes:

- Todos los char y short int se convierten a int. Todos los float a double.
- Para todo par de operandos, lo siguiente ocurre en secuencia:
 - o Si uno de los operandos es un long double, el otro se convierte en long double.
 - o Si uno de los operandos es double, el otro se convierte a double.
 - o Si uno de los operandos es long, el otro se convierte a long.
 - o Si uno de los operandos es unsigned, el otro se convierte a unsigned.



Tipos de Datos

Conversion entre tipos de datos

Ejemplo

```
char ch;  
int i;  
float f;  
double d;
```

```
(ch / i) + ( f + d)    -( f    + i)
```

```
Int  int  double double double double
```

```
(int)    + (double) -(double)
```

```
double    double    double
```

```
resultado=double
```



ConversionDatos.cpp



Pontificia Universidad
JAVERIANA
Colombia

Tipos de Datos

Conversión forzada de tipos de datos.

En C++ podemos obligar a la conversión forzada de tipos de datos utilizando **casting** de tipos de datos.

Para hacer esta operación utilizamos

`(<nombre de tipo>)<expresión>`

Ej:

```
cout << (int)(4.9) << endl; //resultado 4
```

```
cout << (int)(4.6+3) << endl; //resultado 7
```

```
cout << (float) (4.6+3) << endl; //resultado 7.6
```



Ejercicios en Clase

```
//Ejercicio 1
#include <iostream>
using namespace std;
int main()
{
    string estudiantes[] = {"pedro","juan","miguel","sara"};
    int limite =(sizeof(estudiantes)/sizeof(estudiantes[0]));
    for (int i = 0; i < limite; i++)
    {
        cout<<estudiantes[i]<<endl;
    }
}
```



Ejercicio1.cpp



Pontificia Universidad
JAVERIANA
Colombia

Ejercicios en Clase

```
//Ejercicio 2
#include <iostream>
using namespace std;
int main()
{
    int tabla2 [2][3] = { {50,51,52},{53,54,55}};
    for (int n=0;n<2;n++){
        for (int m=0;m<3;m++){
            cout << " ["<<n<<","<< m <<"]:"<<tabla2[n][m];
        }
        cout << endl;
    }
}
```



Ejercicio2.cpp



Pontificia Universidad
JAVERIANA
Colombia

Ejercicios en Clase

```
#include <string>
#include <iostream>
using namespace std;
int main(){
int unsigned w=-100;
int unsigned x=100;
int signed y=100;
int signed z=-100;
int a=100;
cout << (int)(4.9) << endl; //resultado 4
cout << (int)(4.6+3)<< endl; //resultado 7
cout << (float) (4.66+3)<< endl; //resultado 7.6
cout << "w:"<<w<<endl;//Genera error en ejecucion
cout << "X:"<<x<<endl;
cout << "Y:"<<y<<endl;
cout << "Z:"<<z<<endl;
cout << "A:"<<a<<endl;
return 0;
}
```



Ejercicio3.cpp



Ejercicios en Clase



Ejercicio4.cpp



Pontificia Universidad
JAVERIANA
Colombia

Ejercicios en Clase

```
#include <iostream>
using namespace std;
int main()
{
    int tabla2 [2][3] = { {50,61,52},{53,14,55}};
    int mayor, menor, posmMayor, posnMayor, posnMenor, posmMenor;

    mayor=tabla2[0][0];
    posmMayor=0;
    posnMayor=0;

    menor=tabla2[0][0];
    posmMenor=0;
    posnMenor=0;

    for (int n=0;n<2;n++){
        for (int m=0;m<3;m++){
            cout << " ["<<n<<","<<m<<"]:"<<tabla2[n][m];
            if (tabla2[n][m]> mayor ){
                mayor=tabla2[n][m];
                posnMayor=n;
                posmMayor=m;
            }
            if (tabla2[n][m]< menor ){
                menor=tabla2[n][m];
                posnMenor=n;
                posmMenor=m;
            }
        }
        cout << endl;
    }
    cout<< "El numero mayor es :"<<mayor<< " En la posicion ["<<posnMayor<<"]["<<posmMayor<< "]"<<endl;
    cout<< "El numero menor es :"<<menor<< " En la posicion ["<<posnMenor<<"]["<<posmMenor <<"]"<<endl;
}
```



Ejercicio5.cpp



Ejercicios Varios



ErrorLecturasChar.cpp



ErrorLecturasString.cpp



numeroAChar.cpp



kbhit_sleep.cpp



Triqui.cpp



Pontificia Universidad
JAVERIANA
Colombia

Taller 1

La librería Nacional quiere almacenar la información de sus libros en memoria. Nos indica que no tienen mas de 100 libros.

De cada libro se necesita almacenar la siguiente información:

- autor
- nombre;
- isbn;
- fechaPub;
- edicion;
- valor;

De cada autor se necesita el nombre y la nacionalidad.

Al finalizar la carga de información el programa debe mostrar la información de todos los libros así:

- Nombre del autor
- Nacionalidad
- Nombre del libro
- ISBN
- fecha publicación
- Edición
- Valor
- IVA
- Valor Total

Nota: El IVA es el 19% del valor del libro

Ingreso de información

```
Numero de libros a registrar?: 1
Autor:orlando
Nacionalidad del Autor:colombiano
Nombre del libro:La Ballena Azul
ISBN:123564
Fecha publicacion:2022/01/25
Edicion:Segunda
Valor:100000
```

Salida de información

```
Autor:orlando
Nacionalidad del Autor:colombiano
Nombre del libro:La Ballena Azul
ISBN:123564
Fecha publicacion:2022/01/25
Edicion:Segunda
Valor:100000
IVA:19000
Total:119000
```



Escriba un programa en C++ que lea por consola los valores para llenar una matriz de 3x3, luego divida los números de la matriz por el valor en la posición[1][1] y el resultado lo almacene en otra matriz. Al final mostrar las dos matrices

```
Digite valor de la posicion [0][0]:1
Digite valor de la posicion [0][1]:2
Digite valor de la posicion [0][2]:3
Digite valor de la posicion [1][0]:4
Digite valor de la posicion [1][1]:5
Digite valor de la posicion [1][2]:6
Digite valor de la posicion [2][0]:7
Digite valor de la posicion [2][1]:8
Digite valor de la posicion [2][2]:9
```

Matriz original

```
[1][2][3]
[4][5][6]
[7][8][9]
```

Matriz resultado

```
[0.2][0.4][0.6]
[0.8][1][1.2]
[1.4][1.6][1.8]
```

