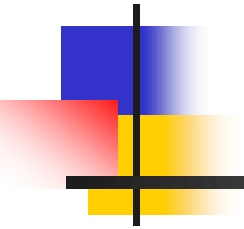


PROGRAMACIÓN ESTRUCTURADA

DESCRIPCIÓN DEL CURSO Y JUSTIFICACIÓN DEL CURSO

- Programación Estructurada es importante para la formación de los estudiantes de Ingeniería de Software y Ciencias de la Computación, ya que les permitirá desarrollar programas de cómputo básicos utilizando estructuras de secuencia, selección e iteración; así como llamadas y creación de subrutinas; con base en el procesamiento algorítmico. Esta asignatura tiene como propósito, aportar los paradigmas y las técnicas principales de programación para su aplicación en el diseño, desarrollo e implantación de sistemas computacionales.

COMPETENCIA DE LA ASIGNATURA

- Desarrolla aplicaciones computacionales eficientes, fundamentado en las metodologías del paradigma de la programación estructurada.

CONTENIDO

UNIDAD 1
PROGRAMACIÓN ESTRUCTURADA

COMPETENCIA:

Identifica los elementos básicos del paradigma de la programación estructurada, para el desarrollo de programas computacionales, de manera eficiente.

SECUENCIA DE CONTENIDOS:

1. Introducción a la Programación Estructurada.
2. Estructura general de un programa.
3. Operadores y expresiones.

CONTENIDO

UNIDAD 2
ESTRUCTURAS DE CONTROL DE FLUJO

COMPETENCIA:

Desarrolla programas computacionales, utilizando las estructuras de control de flujo de la programación estructurada.

SECUENCIA DE CONTENIDOS:

1. Funciones de Entrada/ Salida.
2. Estructuras selectivas de control de flujo de un programa.
3. Estructuras repetitivas de control de flujo de un programa.

CONTENIDO

UNIDAD 3
MODULARIDAD

COMPETENCIA:

Desarrolla programas computacionales en forma modular, mediante el empleo de funciones.

SECUENCIA DE CONTENIDOS:

1. Introducción a la modularidad.
2. Funciones definidas por el usuario.
3. Paso de parámetros a funciones.
4. Ámbito de las variables y clases de almacenamiento.
5. Funciones de biblioteca.

CONTENIDO

UNIDAD 4
ARREGLOS

COMPETENCIA:

Desarrolla programas computacionales que requieren del manejo de conjuntos de datos, mediante el uso de la estructura arreglo.

SECUENCIA DE CONTENIDOS:

1. Introducción a las estructuras de datos.
2. La estructura arreglo.
3. Cadenas.

CONTENIDO

UNIDAD 5
ESTRUCTURAS REGISTROS Y APUNTADORES

COMPETENCIA:

Desarrolla programas computacionales que requieren del manejo de conjuntos de datos, mediante el empleo de estructuras dinámicas de datos.

SECUENCIA DE CONTENIDOS:

1. La estructura registro.
2. Apuntadores.
3. Asignación dinámica de la memoria.

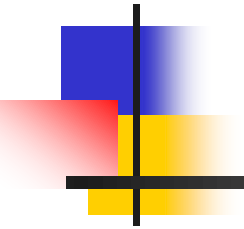
Unidad 1.- Programación Estructurada

COMPETENCIA:

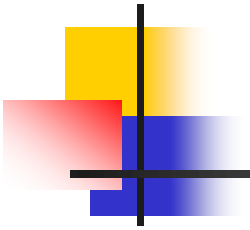
Identifica los elementos básicos del paradigma de la programación estructurada, para el desarrollo de programas computacionales, de manera eficiente.

SECUENCIA DE CONTENIDOS:

1. Introducción a la Programación Estructurada.
2. Estructura general de un programa.
3. Operadores y expresiones.



1.Introducción a la Programación Estructurada



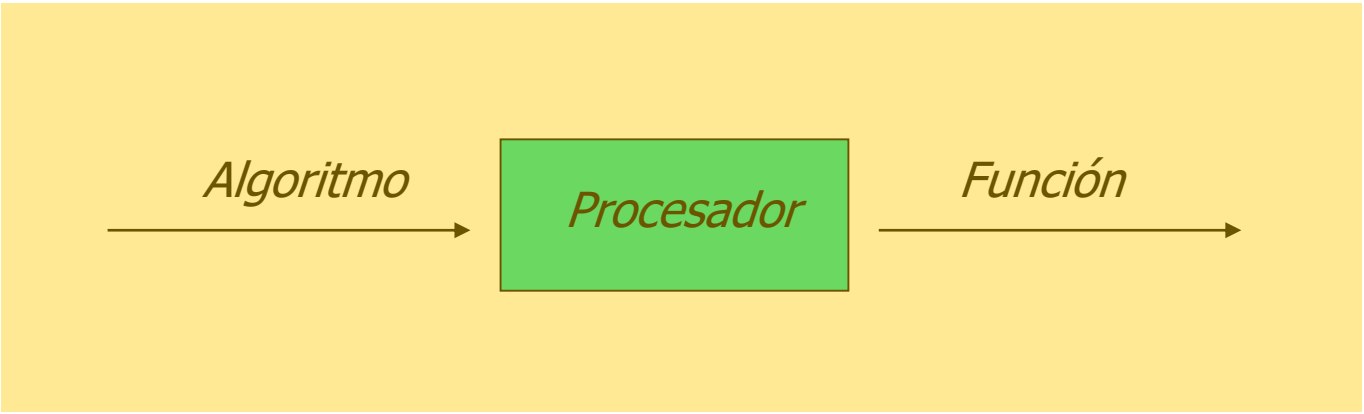
Lenguajes de programación

- La forma en que nos comunicamos influye en la manera en la que pensamos, y viceversa.
  - Un lenguaje es un conjunto sistemático de reglas para comunicar ideas.

- La forma en la que programamos las computadoras influye en lo que pensamos de ellas.
  - Lenguaje de Programación es un sistema notacional para describir cómputos de manera entendible para la máquina y el hombre.
  - Es un lenguaje usado por una persona para expresar un proceso mediante el cual la computadora resolverá un problema.

Lenguajes de programación

- Para que un procesador realice un proceso se le debe suministrar un algoritmo. El procesador debe ser capaz de interpretar el algoritmo, lo que significa:
  - Comprender las instrucciones de cada paso.
  - Realizar las operaciones correspondientes.



Lenguajes de programación

- Cuando el procesador es una computadora, el algoritmo se expresa en un formato que se denomina ***programa,*** un programa se escribe en un ***lenguaje de programación*** y las operaciones que conducen a expresar un algoritmo en un lenguaje de programación se llaman ***programación***.

- Así pues, los lenguajes utilizados para escribir programas de computadoras son los ***lenguajes de programación***.

- Tipos de lenguajes de programación:
  - Lenguaje máquina
  - Lenguajes de bajo nivel
  - Lenguajes de alto nivel

Tipos de lenguajes de programación

- Lenguaje máquina:
  - Están escritos en lenguajes directamente inteligibles por la computadora, ya que sus instrucciones son cadenas binarias. También se le conoce como código binario.

Dirección	*Contenido*			
0100	0010	0000	0000	0100
0101	0100	0000	0000	0101
0102	0010	0000	0000	0110
.	.	.	.	.
.	.	.	.	.

Tipos de lenguajes de programación

- Lenguajes de bajo nivel:
  - Son más fáciles de utilizar que los lenguajes de máquina, pero, al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el ensamblador. Las instrucciones en leguaje ensamblador son conocidas como nemotécnicos.

```
ADD     M,      N,      P
```

  - El programa original escrito en lenguaje ensamblador se denomina ***programa fuente*** y el programa traducido en lenguaje máquina se conoce como ***programa objeto***. El traductor de programas fuente a objeto es un programa llamado ***ensamblador***.

Tipos de lenguajes de programación

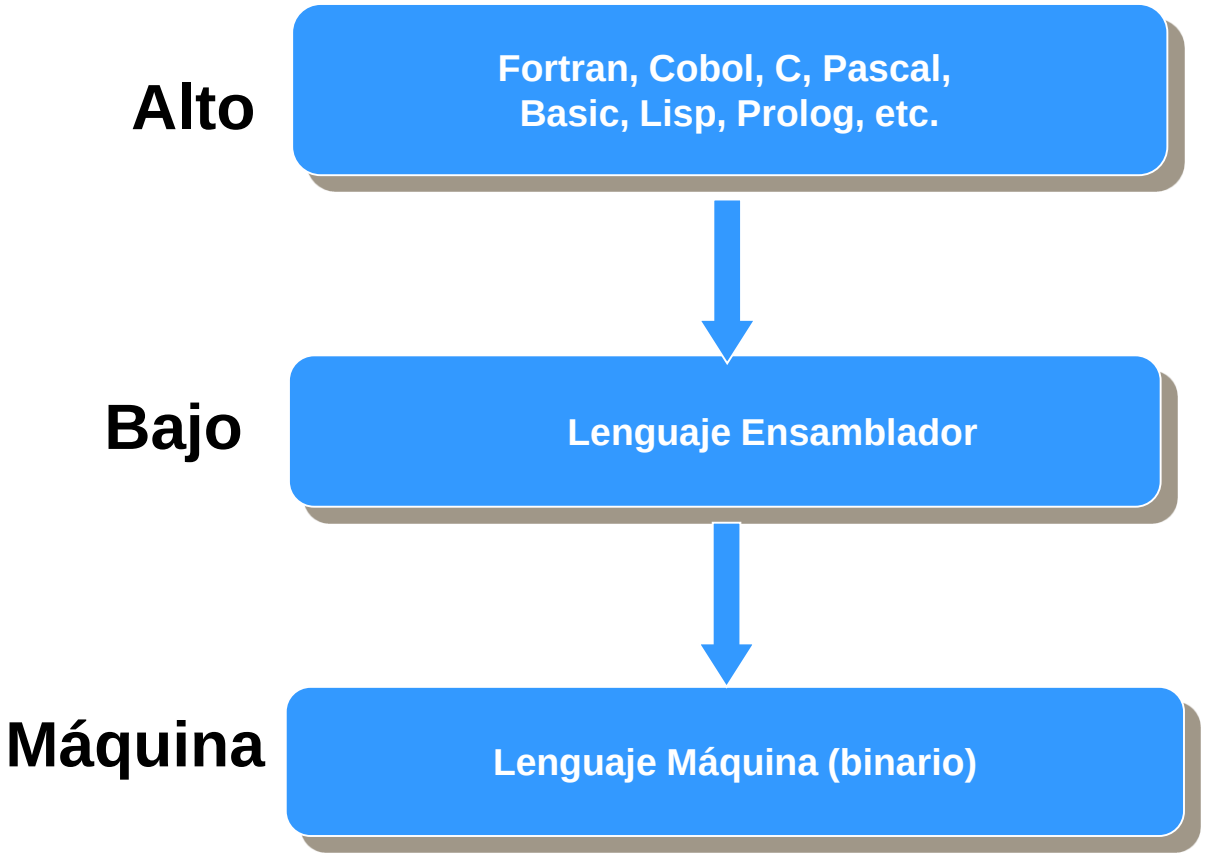
- Lenguajes de alto nivel:
  - Son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores.
  - Al igual que los programas ensambladores los programas fuente tienen que ser traducidos por programas traductores, llamados compiladores e intérpretes.

Tipos de lenguajes de programación

- Lenguajes de alto nivel:
  - Son los más usados por los programadores.
  - Facilitan la programación, son independientes de la computadora, son portables, tiempo de programación más corto, Se usan reglas sintácticas parecidas a los lenguajes humanos: READ, WRITE, OPEN,.., Programas más económicos.
  - Se limitan en cuanto al uso de los recursos internos de la máquina, consumen más recursos: memoria, son más lentos cuanto de más alto nivel son.

- Los lenguajes de 4 y 5 generación son más parecidos a como se expresan los humanos: imágenes o símbolos, frases en lenguaje natural, etc.

Tipos de lenguajes de programación

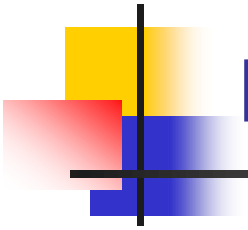


Herramientas básicas de programación

- Una herramienta de programación es un programa informático que usa un programador para crear, depurar, gestionar o mantener un programa.

Herramientas básicas de programación

- Sistema de seguimiento de errores: Bugzilla
- Generador de código: Make
- Conversor de código: JTest
- Compilador: gcc
- Depurador: gdb
- Desensamblador: OllyDbg
- Generador de documentacion: Javadoc
- Generador de GUI: NetBeans cuenta con uno.
- Detector de fugas de memoria: dmalloc
- Analizador sintáctico: Lex

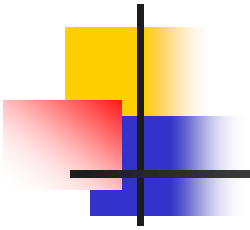


Herramientas básicas de programación

- Profilers: Java Virtual Machine Tools Interface (JVM TI)
- Control de versiones: CVS
- Herramienta de busqueda: grep
- Entorno de desarrollo integrado: Zinjai
- Generador de estilo: indent
- Editor de texto: jEdit

Proceso de elaboración de un programa

1. Entender el problema.
2. Hacer el análisis del mismo.
3. Programar el modelo de solución (Algoritmos).
4. Codificarlo.
5. Cargarlo a la computadora para ejecución y ajuste.
6. Documentación del programa.
7. Mantenimiento a lo largo de su vida útil.



Proceso de elaboración de un programa

- Entender el problema:
  - Se trata de crear y mantener una idea clara, un "mapa mental" del problema propuesto y de ser capaz de abarcarlo de un solo vistazo.
  - No se toman en cuenta detalles y particularidades operativas en primera instancia.
  - La descripción debe procurar incluir los aspectos más relevantes a tomar en cuenta.

Proceso de elaboración de un programa

- Analizar el problema:
  - ¿Qué debe hacer el programa?
  - Consiste en detallar las partes que intervienen en el problema de manera que se pueda proponer un modelo de solución. Puede estar basado en los datos que maneja el sistema y la manera como fluyen por él. Se describe como una caja negra.
  - El resultado puede ser *diagramas o bosquejos que muestren el flujo de información y las funciones que desempeñan*.
  - También puede ser una descripción de cómo se propone que funcione el sistema o programa, Indicando: **entradas, salidas y proceso a realizar.**

Proceso de elaboración de un programa

- Programar el modelo de solución:
  - Programa: es un conjunto de declaraciones de **estructuras de datos**, seguidas de un conjunto de proposiciones (**estructuras de control** en general) que siguen ciertas reglas de construcción.
    - **Estructuras de control:** son las formas que existen para dirigir el flujo de acciones que el procesador efectuará sobre los datos.
    - **Estructuras de datos**: son las diversas maneras en que se pueden organizar los datos para facilitar la representación de objetos.

Proceso de elaboración de un programa

- Programar el modelo de solución:
  - ¿Cómo se logra hacer la tarea?
  - Se procede a convertir un sistema en programa de computadora escrito en seudocódigo o diagramas de flujo.
  - Se aplican reglas y elementos estructurales bien definidos para construir los programas.
  - Podemos imaginar este proceso como la creación de la estructura de un edificio.
  - NO se maneja un lenguaje de programación en particular.

Proceso de elaboración de un programa

- Codificación:
  - Como resultado del paso anterior ya tenemos un modelo propuesto escrito en seudocódigo.
  - La codificación de un programa, consiste en traducir la solución en seudocódigo a algún lenguaje de programación, el cual pueda ser ejecutado en la computadora.
  - Debemos de estar familiarizados con los elementos y forma de codificar del lenguaje que vayamos a utilizar y emplear el más adecuado para cada tipo de problemática.

Algunos autores hacen la distinción entre programar y codificar, para diferenciar claramente estas etapas.

Proceso de elaboración de un programa

- Ejecución y ajuste (pruebas):
  - Una vez codificado el programa y *compilado*, se puede ejecutar y probar en la computadora. Pueden darse los siguiente errores:
    - **Error de sintaxis o de compilación:** se refiere a errores en el uso del lenguaje de programación.
    - **Errores de ejecución:** al momento de correr el programa se pueden dar divisiones por cero, raíces de números negativos, etc. El programa se detiene generalmente.
    - **Errores de lógica de programación:** son más graves, pues indican que no se encontró una solución adecuada al problema y el programa no obtiene lo que se esperaba de él. El algoritmo no esta bien diseñado.
  - Se recomienda dedicar más tiempo a las etapas de análisis y programación para tener menos problemas en la etapa de ajuste y depuración.

Proceso de elaboración de un programa

- Documentación:
  - Documentación técnica:
    - Describen cuestiones técnicas del lenguaje para facilitar su comprensión y actualización en el futuro.
    - Es la recopilación del trabajo desarrollado en las etapas anteriores.
    - Está enfocado al programador.
  - Documentación del usuario:
    - Destinada al usuario para ayudarlo a manejar el sistema, debe de estar libre de tecnicismos en lo posible.

Proceso de elaboración de un programa

- Mantenimiento:
  - Cuando el sistema tendrá un uso prolongado, lo más probable es que requiera revisiones y adecuaciones para satisfacer las necesidades cambiantes de los usuarios.

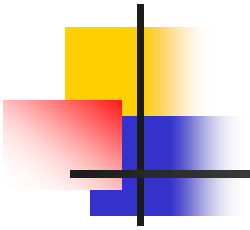
  - Un buen programa es:
    - Claro, flexible, admite cambios y mejoras posteriores, es adaptable y robusto.

Proceso de elaboración de un programa

- El proceso de elaboración de programas puede ser tan simple o complejo como el problema mismo a resolver.

- Existen varias metodologías y técnicas que ayudan a los programadores generar programas de una manera más precisa y productiva.

- Aunque en esencia la programación es un proceso creativo – en sus inicios se le consideraba un arte más que una ciencia-, se pueden definir fases o etapas que todos los programadores deben seguir para desarrollar programas de una forma más eficiente.

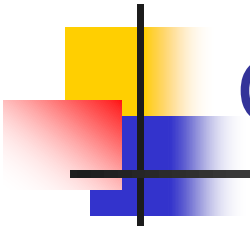


Proceso de elaboración de un programa

- La adopción de una metodología de programación proporciona múltiples beneficios a los programadores:
  - Un proceso definido y eficiente.
  - Facilita la tarea de creación, revisión y depuración de programas.
  - Un proceso repetible.

Compiladores e intérpretes

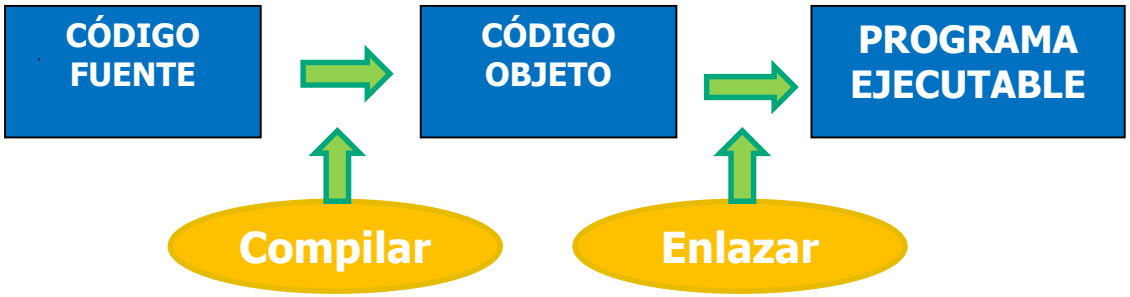
- Traductores:
  - Son programas que traducen los programas escritos en un lenguaje en cuestión y que los ejecuta directamente, o los transforma en una forma adecuada para su ejecución. Los traductores pueden ser:
    - Intérpretes:
      - Son traductores que toman un programa fuente, lo traducen y a continuación lo ejecutan línea a línea. No se produce un resultado físico (código máquina), sino lógico (ejecución).
    - Compiladores:
      - Son programas que traducen los programas fuentes en una forma adecuada para su ejecución.



Compiladores e intérpretes

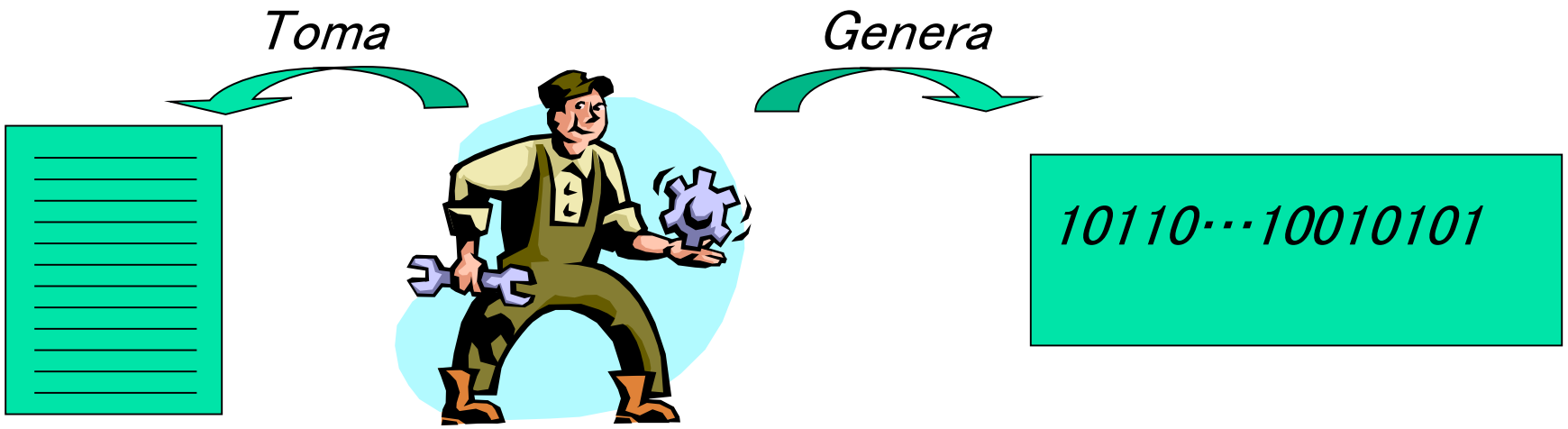
- Ventajas:
  - Código compilado es más rápido.
  - El código interpretado no depende de la plataforma en la cual se creó originalmente.
- Desventajas:
  - El programa compilador es dependiente al sistema operativo o plataforma.
  - El código interpretado suele ser menos rápido que uno compilado.

Proceso de compilación



- **CÓDIGO FUENTE**: Es el programa que nosotros escribimos en un editor.
- **CÓDIGO OBJETO**: Es el programa fuente pero traducido a lenguaje máquina (sucesión de ceros y unos).
- **PROGRAMA EJECUTABLE**: Es el programa objeto más las librerías del lenguaje, (en lenguaje máquina) se graba con la extensión .EXE. y no necesita el programa que hemos utilizado para crearlo, para poder ejecutarlo.
- **COMPILAR**: Es un proceso en el que se revisa que el programa este correctamente escrito y se traduce a código objeto.
- **ENLAZAR**: Es el proceso en el que se agregan las utilerías y librerías para armar un sólo programa que pueda ser ejecutado.

Proceso de compilación



Proceso de compilación

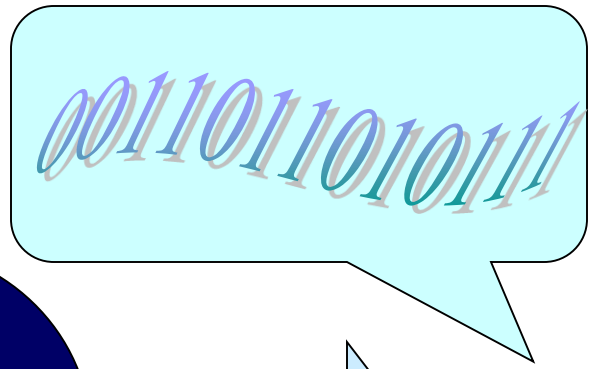
Lenguaje Programación

```
#include <stdio.h>

main()
{
int i,j,k;

i=4;
j=3;

  for (i=0; i<100; i++)
  {
    k= i*i;
    printf("cuadrado de %d =
          %d", i, K);
 }
exit(o);
}
```

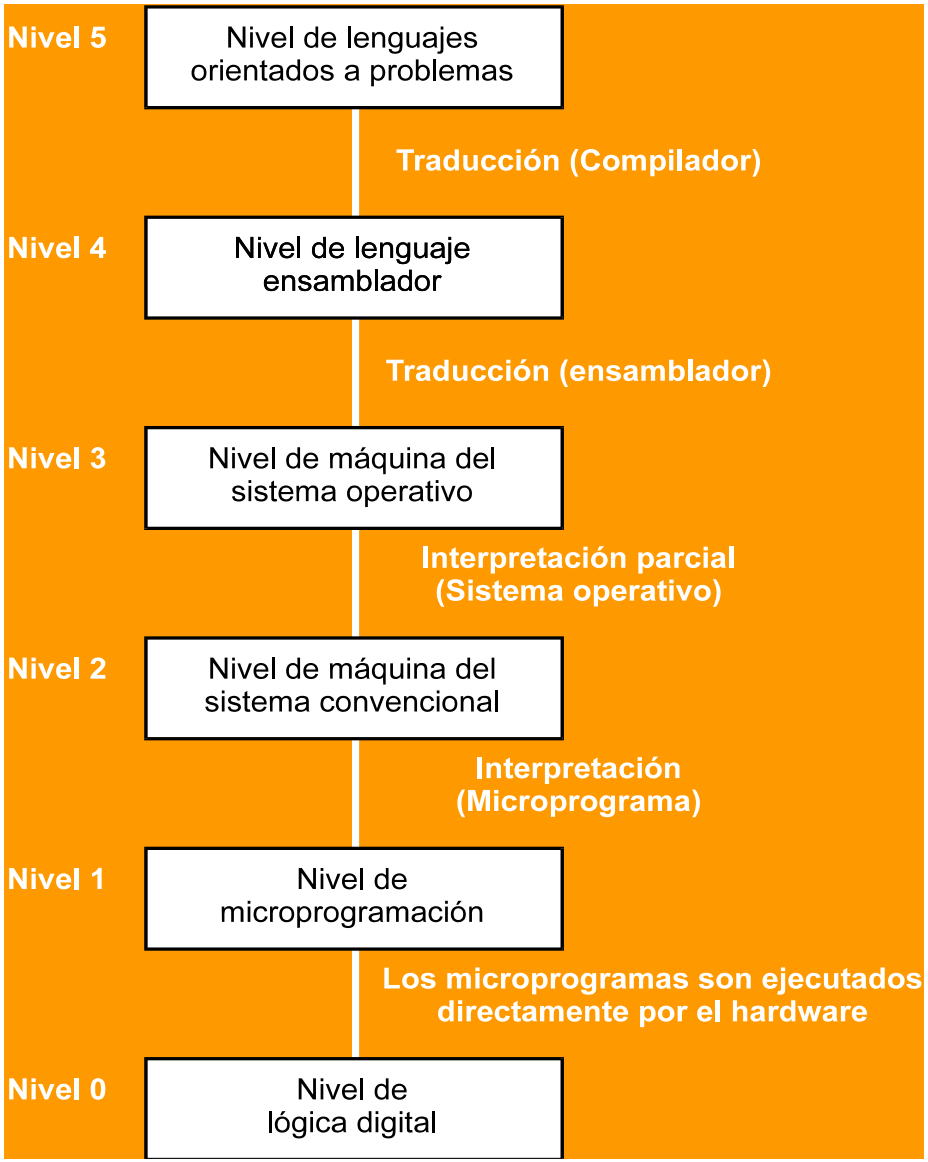
CÓDIGO FUENTE

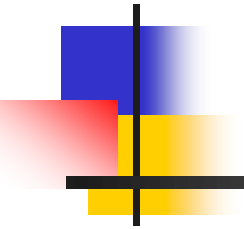
Compilador

CÓDIGO MÁQUINA



Máquina Multinivel Actual





2. Estructura general de un programa

Estructura general de un programa en C

Programación en C - Historia

- Muchas ideas provienen de BCPL (Martin Richards, 1967) y de B (Ken Thompson, 1970).
- C fue diseñado originalmente en 1972 para el SO UNIX en el DEC PDP-11 por Dennis Ritchie en los Laboratorios Bell.
- Primer Libro de referencia de C: The C Programming Language (1978) de Brian Kernighan y Dennis Ritchie.
- En los 80, gran parte de la programación se realiza en C.
- En 1983 aparece C++ (orientado a objetos).
- En 1989 aparece el estándar ANSI C.
- En 1990 aparece el estándar ISO C (actual estándar de C). WG14 se convierte en el comité oficial del estándar ISO C.

Estructura general de un programa en C

- *Directivas del preprocesador*

```
#include <stdio.h>
```

- *Macros del preprocesador*
- *Declaraciones globales*
  - *Prototipos de funciones*
  - *variables*
- *Función principal* ***main***

```
main()
{
    Declaraciones locales
    Sentencias
}
```

- *Definiciones de otras funciones*

```
/* Programa Saludo.c – Programa de saludo */
#include <stdio.h>
/* Este programa imprime: Bienvenido a C */
        int main()
        {
                printf("Bienvenido a C\n");
                return 0;
        }
```

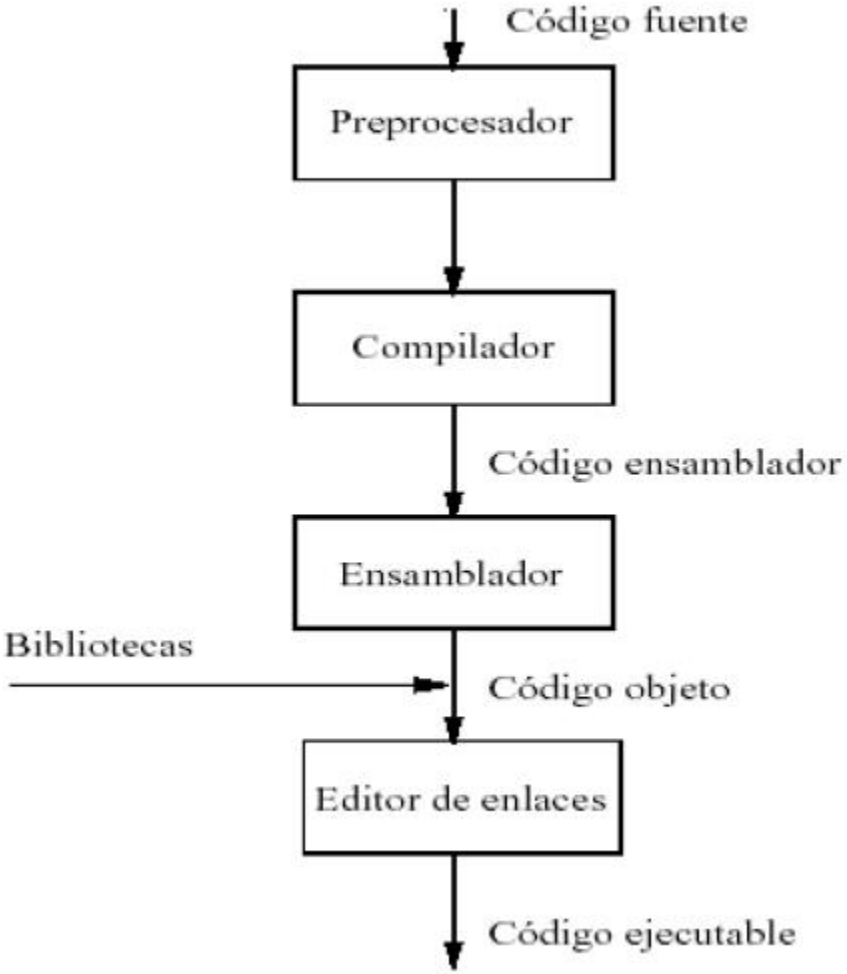
```
int main()
{
        printf("Bienvenido a C\n");
        return 0;
}
```

Estructura general de un programa en C

Directivas del preprocesador

- Las directivas son instrucciones al compilador antes de que se compile el programa principal.
- Pueden definir macros, nombres de constantes, archivos fuente adicionales, pero principalmente archivos de cabecera.
- Directivas más usuales:
  - #include
    - Indica al compilador que lea el archivo fuente que viene a continuación de ella y su contenido lo inserte en la posición donde se encuentra dicha directiva.
      - #include<stdio.h>
  - #define
    - Indica al procesador que defina un item de datos u operación para el programa C.
      - #define JORNADA_TRABAJO 40

Proceso de generación de un ejecutable



Crear un archivo con extensión **.c** con un editor de texto.

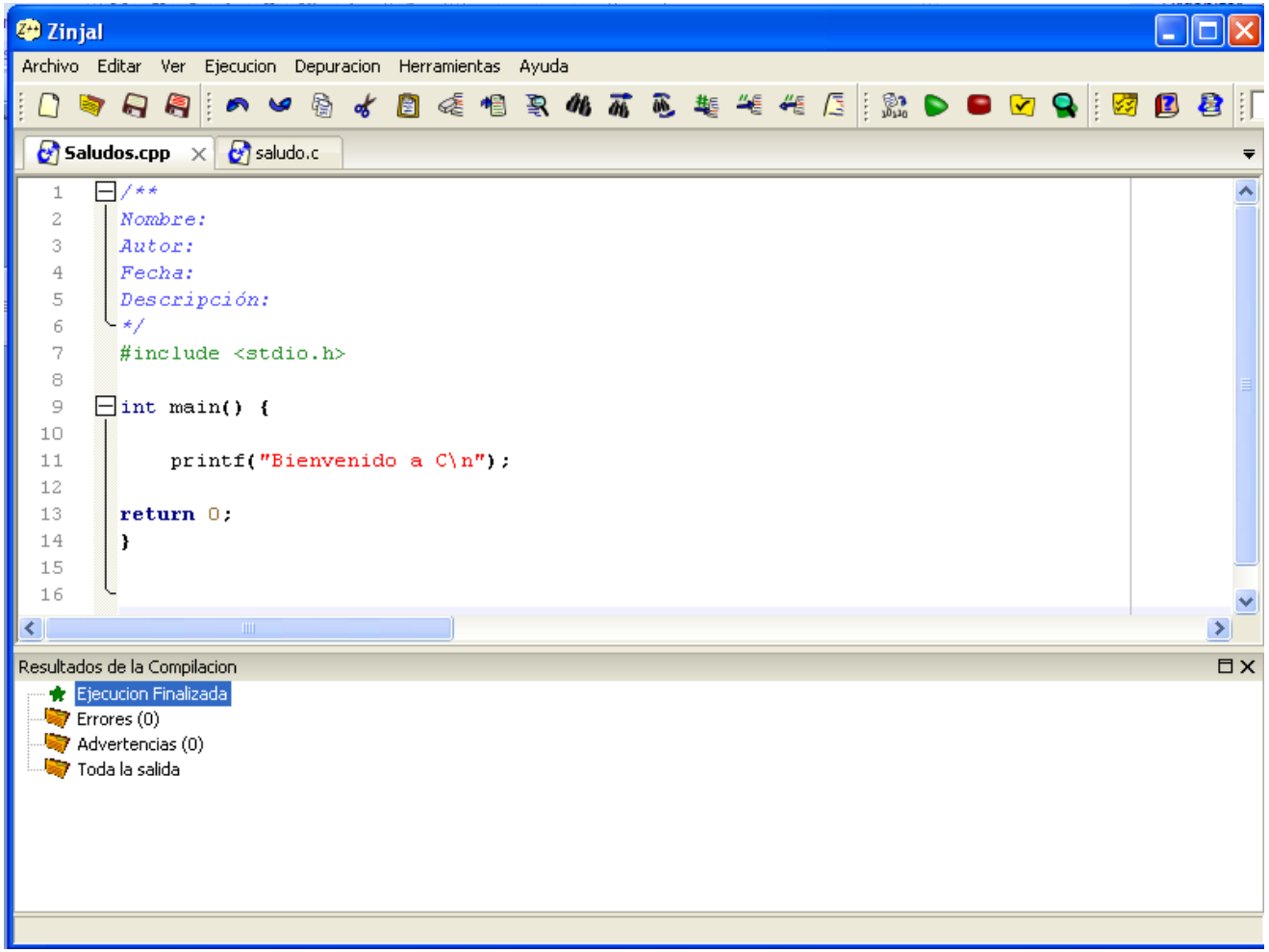
Compilación básica: ***gcc – c saludo.c***
genera el código objeto ***saludo.o***

El mandato ***gcc saludo.o -o saludo*** genera el archivo ejecutable ***saludo***

El programa se ejecuta tecleando ***saludo***

Mediante un IDE (Entorno de Desarrollo Integrado)

- Proporciona las herramientas para editar, compilar, depurar y ejecutar programas.



Estructura general de un programa en C

Características de C

- Lenguaje de programación de propósito general, muy adecuado para programación de sistemas (unix fue escrito en C).
- Lenguaje relativamente pequeño: solo ofrece sentencias de control sencillas y funciones.
- La E/S no forma parte del lenguaje, sino que se proporciona a través de una biblioteca de funciones.
- Permite la agrupación de instrucciones. Programación estructurada.
- Permite la separación de un programa en módulos que admiten compilación independiente. Diseño modular.
- Programas portables.

Estructura general de un programa en C

Inconvenientes y cuidados a considerar

- Sin una programación metódica puede ser propenso a errores difíciles de encontrar.
- La versatilidad de C permite crear programas difíciles de leer.

```
#define _ -F<00||--F-OO--;
int F=00,OO=00;
main(){F_OO();printf("%1.3f\n",4.*-
   F/OO/OO);}F_OO()
{.....}
```

- Se recomienda escribir código legible y respetar estándares de codificación.
- Recordemos que CODIFICAR es traducir a un lenguaje de programación el resultado de generar algoritmos.

Elementos de un programa en C

- Identificadores
- Palabras reservadas
- Comentarios
- Signos de puntuación
- Archivos de cabecera

Elementos de un programa en C

Identificadores

- Un ***identificador*** es una secuencia de caracteres, letras, dígitos y subrayados (_). El primer carácter debe ser una letra. C es sensible al texto.
  - Ejemplo:

    Salario_empleado, empleado1, Suma

    fecha_compra_casa, A, i

- Los identificadores pueden tener cualquier longitud, pero sólo son significativos los 32 primeros.

- Los identificadores no pueden ser palabras reservadas como ***if***, ***switch***, ***else***, etc.

Elementos de un programa en C

Palabras reservadas

- Una ***palabra reservada*** (keyword o reserved word), es una característica del lenguaje C asociada con algún significado especial. Una palabra reservada no se puede utilizar como nombre de un identificador o función.

  - Ejemplo:

Break	***sizeof***	***do***
Case	***struct***	***else***
Const	***switch***	***int***

Elementos de un programa en C

/* Comentarios */

- Los comentarios se incluyen en los programas para proporcionar explicaciones a los lectores.
  - Ejemplo:
    - /* ejemplo1.c - Primer programa en C */

```
/* Programa: ejemplo1.c
   Programador: Alfonso XIII
   Descripción: Primer programa en C
   Fecha creación: septiembre 2006
   Revisión: Ninguna */
```

Elementos de un programa en C

Signos de puntuación y separadores

- Todas las sentencias deben terminar con punto y coma (;). Otros signos de puntuación son:

!	%	^	*	(	)	=
.	|	<	>	?	"	´

- Los separadores son:
  - Espacios en blanco
  - Tabulaciones
  - Retornos de carro
  - Avances de línea

Elementos de un programa en C

Archivos de cabecera

- Es un archivo especial que contiene declaraciones de elementos y funciones de la biblioteca.

- Para utilizar macros, constantes, tipos y funciones almacenadas en una biblioteca, un programa debe utilizar la directiva ***#include*** para insertar el archivo de cabecera correspondiente.

  - Ejemplo:

    - #include <stdio.h>
    - #include "stdio.h"

Tipos de datos en C

- Enteros
- Números en coma flotante (reales)
- Caracteres

Tipos de datos en C

Enteros (int)

- Los enteros son adecuados para aplicaciones que trabajen con datos numéricos.

TIPOS DE DATOS ENTEROS

Tipo	Ejemplo	Tamaño en bytes	Rango Mínimo … Máximo
short	-15	2	-128 … 127
int	1024	2	-32768 … 32767
long	22144	4	-2147483648 … 2147483637
unsigned int	42325	2	0 … 65535
short int	100	2	-128 … 127
unsigned long			0 … +4294967295
unsigned short			0 … +255

Tipos de datos en C

Tipos de coma flotante (float / double)

- Los tipos de datos coma (punto) flotante representan números reales que contienen una coma (un punto) decimal, tal como 3.14159, o números muy grandes, tales como 1.85×10^{15}.

TIPOS DE DATOS EN COMA FLOTANTE (BORLAND C)

Tipo	Ejemplo	Tamaño en bytes	Rango Mínimo	Rango Máximo
float	10.5	4	3.4×10^{-38}	3.4×10^{38}
double	0.00045	8	1.7×10^{-308}	1.7×10^{308}
long double	1e-8	8	*igual que double*	

Caracteres (char)

- Un carácter es cualquier elemento de un conjunto de caracteres predefinidos o alfabeto. La mayoría de las computadoras utilizan el conjunto de caracteres ASCII.

TIPOS DE DATOS CHAR

Tipo	Ejemplo	Tamaño en bytes	Rango Mínimo … Máximo
char	C	1	0 … 255

- Internamente los caracteres se almacenan como números. La letra ***A*** por ejemplo, se almacena internamente como el número ***65***, la letra ***B*** es ***66***, etc.

- Puesto que los caracteres se almacenan internamente como números se pueden realizar operaciones aritméticas con datos tipo char. Por ejemplo, se puede convertir una letra minúscula ***a*** a una letra mayúscula ***A***, restando 32 del código ASCII:

```
char car_uno = 'a';
….
Car_uno = car_uno – 32;
```

Tamaño de Variables (*sizeof*)

- En muchos programas es necesario conocer el tamaño (cantidad de bytes) que ocupa una variable, por ejemplo en el caso de querer reservar memoria para un conjunto de ellas.

- El tamaño es dependiente del compilador que se use, lo que producirá, un problema si luego se quiere compilar el programa con un compilador distinto del original.

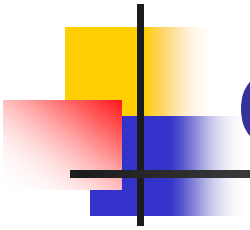
- Para salvar este problema y mantener la portabilidad, es conveniente que cada vez que haya que referirse al TAMAÑO en bytes de las variables, se haga mediante un operador llamado "**sizeof**" que calcula sus requerimientos de almacenamiento.

- También esta permitido el uso de sizeof con un tipo de variable, es decir:

  sizeof(int)
  sizeof(char)
  sizeof(long double)

Constantes en C

- Constantes literales
- Constantes definidas
- Constantes declaradas



Constantes en C

Constantes literales

- Se escriben directamente en el texto del programa y pueden ser:
  - Constantes enteras
  - Constantes reales
  - Constantes carácter
  - Constantes de cadena

Constantes en C

Constantes literales

- Constantes enteras
  - No utilizar comas ni otros signos de puntuación en números enteros o completos: 12345 en lugar de 123.45
  - Para forzar un valor al tipo long, terminar con una letra L o l: 1024L
  - Para forzar un valor al tipo unsigned, terminar con letra U: 3457U
  - Para representar un entero en octal, se debe estar precedido de 0: 0777
  - Para representar un entero en hexadecimal, se debe estar precedido por 0X: 0XFF3A

Constantes en C

Constantes literales

- Constantes reales
  - Representa un número real.
  - Siempre tiene signo y representa aproximaciones en lugar exacto de valores exactos.
  - Ejemplos:
    - 82.345, .63, 83., 47e-4, 1.25E7, 61.e+4
    - 2.5E4 equivale a 25000
    - 5.435E-3 equivale a 0.005435
  - Existen tres tipos de constantes:
    - float (4 bytes)
    - Double (8 bytes)
    - long double (10 bytes) //puede cambiar por el compilador

Constantes en C

Constantes literales

- Constantes carácter
  - Es un carácter del código ASCII encerrado entre apóstrofes.
  - ‘A’, ‘b’, ‘c’
  - Soporta también caracteres especiales que no se pueden representar utilizando el teclado (códigos ASCII altos y las secuencias de escape).
  - ‘\n’ nueva línea
  - ‘\r’ retorno de carro
  - ‘\t’ tabulación
  - ‘\\’ barra inclinada inversa

Constantes en C

Constantes literales

- Aritmética con caracteres C
  - char c;
  - c = 'T' + 5 /*suma 5 al carácter ASCII */

- En el ejemplo anterior se almacena Y en c, ya que el valor ASCII de T es 84 y al sumarle 5 produce 89 que es el código de la letra Y.
  - int j = 'p'
  - No pone la letra p en j, asigna 80 – ASCII de p – a la variable j.

```
#include <stdio.h>

int main() {
char val1='a';
char val2[]="a";

printf("%d", sizeof(val1));
printf("%d", sizeof(val2));
            // agregar código

return 0;
}
```

Constantes en C

Constantes literales

- Constantes cadenas
  - Es una secuencia de caracteres encerrados entre doble comillas.
  - Ejemplos:
    - “123”, “12 de octubre”, “esto es una cadena”
  - Se representan por una serie de caracteres ASCII más un 0 o nulo (NULL).
  - El carácter nulo marca el final de la cadena y se inserta automáticamente al final de las constantes de cadenas.

Constantes en C

Constantes literales

- Diferencias entre constantes carácter y constantes cadenas: ‘z’ y “z”.
- ‘z’ es una constante de carácter simple con longitud 1, “z” es una constante de cadena de caracteres también con longitud 1.
- La diferencia es que la constante de cadena incluye un cero (nulo) al final de la cadena, ya que C necesita saber dónde termina la cadena, la constante carácter no incluye nulo, ya que se almacena como un entero.

Constantes en C

Constantes definidas (simbólicas)

- Las constantes pueden recibir nombres simbólicos mediante las directiva #define.
  - Ejemplo:
    - #define NUEVALINEA \n
    - #define PI 3.14159
    - #define VALOR 54
- C sustituye los valores \n, 3.14159 y 54 cuando se encuentran las constantes simbólicas NUEVALINEA, PI y VALOR.
- El compilador lo que hace es sustituir en el programa todas las ocurrencias de VALOR por 54, antes de realizar sintácticamente el programa fuente.

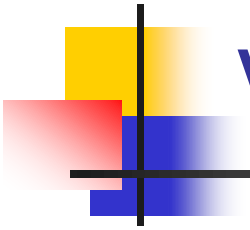
Constantes en C

Constantes declaradas (const, volatile)

- El cualificador ***const*** especifica que el valor de una variable no se puede modificar durante el programa. Cualquier intento de modificar el valor de la variable definida con ***const*** producirá un mensaje de error.
  - Notación: *const* tipodato nombreConstante = valorConstante;
    - Ejemplo:

      const char CARÁCTER = '@';
      const int MESES = 12;

- La palabra reservada volatile actúa como const, pero su valor puede ser modificado, no solo por el propio programa, sino también por el hardware o por el software del sistema. Las variables volátiles, sin embargo, no se pueden guardar en registros, como en el caso de variables normales.



Variables en C

- En C una variable es una posición con nombre en memoria, donde se almacena un valor de un cierto tipo de dato.
- Las variables pueden almacenar todo tipo de datos: cadenas, números y estructuras.
- Típicamente tiene un nombre (identificador) que describe su propósito.
- **Toda variable utilizada en el programa debe ser declarada previamente**.
- La definición en C debe situarse al principio del bloque.
- Al definir una variable se reserva un espacio de almacenamiento en memoria.

Variables en C

- Para crear una variable: ***tipo nombre;***
  - Ejemplo:
    - char respuesta;
    - float HorasPorSemana;
    - short DiaSemana;

- Se puede declarar una variable al principio de un archivo o de un bloque de código, al principio de una función.

- Inicialización de variables, formato general:
  tipo nombre = expresión;

  - Ejemplo:
    - char respuesta = 's';
    - int contador = 1;
    - float peso = 156.45;

Variables carácter (char)

- El lenguaje C guarda los caracteres como números de 8 bits de acuerdo a la norma **ASCII** extendida, que asigna a cada caracter un número comprendido entre 0 y 255. Otros lenguajes como Java asignan dos bytes para representar un carácter (compatible con la norma **UNICODE**).

- Las variables que vayan a alojar caracteres se definen como (**char c;**), sin embargo, también funciona como (**int c ;**). Esta última opción desperdicia un poco más de memoria que la anterior, pero en algunos casos particulares presenta ciertas ventajas por ejemplo para leer un número NO comprendido entre 0 y 255. (por ejemplo, se usa el -1 para el representar el EOF, fin de archivo ó End Of File).

Variables carácter (char)

- Las variables del tipo caracter también pueden ser inicializadas en su definición, por ejemplo es válido escribir:

  char c = 97 ;

- Así c contiene el valor ASCII de la letra "a", sin embargo esto resulta algo engorroso, ya que obliga a recordar dichos códigos . Existe una manera más directa de asignar un carácter a una variable; la siguiente inicialización es idéntica a la anterior :

  char c = 'a' ; // delimitar el caracter con comilla simple

Enumeraciones

- Las constantes enumeradas permiten crear listas de elementos afines.
  - Ejemplo:
    - *enum* Colores {Rojo, Naranja, Amarillo, Verde};
    - *enum* Valor {Verdadero, Falso};
    - *enum* LucesTrafico {Verde, Amarillo=10, Rojo};

- Cuando se procesa esta sentencia, el compilador asigna un valor que comienza en 0 a cada elemento enumerado; rojo equivale a 0, naranja es 1, etc. El compilador enumera los identificadores automáticamente.

Rojo	Naranja	Amarillo	Verde
0	*1*	*2*	*3*

Conversión de tipos de datos

- Con frecuencia se necesita convertir el valor de un tipo a otro sin cambiar el valor que representa. Las conversiones de tipos pueden ser implícitas (ejecutándose automáticamente) o explícitas (solicitadas específicamente). C hace muchas conversiones de tipo automáticamente:
  - C convierte valores cuando se asigna el valor de un tipo a una variable de otro tipo.
  - C convierte valores cuando se combinan tipos mixtos en expresiones.
  - C convierte valore cuando se pasan argumentos a funciones.

Conversión automática de tipos de datos

- Cuando dos ó mas tipos de variables distintas se encuentran dentro de una misma operación o expresión matemática, ocurre una conversión automática del tipo de las variables. En todo momento al realizarse una operación se aplica la siguiente secuencia de reglas de conversión (previamente a la realización de dicha operación):

- Reglas:
  - Las variables del tipo char ó short se convierten en int.
  - Si los operandos tienen diferentes tipos, la siguiente lista determina a qué operación convertirá:

    int
    unsigned int
    long
    unsigned long
    float
    double

    el tipo que viene primero en la lista se convertirá en el que viene segundo.

Conversión Forzada de Tipos de datos (casting)

- Las conversiones automáticas pueden ser controladas a gusto por el programador, imponiendo el tipo de variable al resultado de una operación.

- Supongamos por ejemplo tener:

```
double d , e , f = 2.33 ;
int i = 6 ;
e = f * i ;
d = (int) ( f * i ) ;
```

- En la primer sentencia calculamos el valor del producto (f * i) , que según lo visto anteriormente nos dará un **double** de valor 13.98 , el que se ha asignado a e. Si en la variable d quisiéramos reservar sólo el valor entero de dicha operación bastará con anteponer, encerrado entre paréntesis, el tipo deseado. Así en d se almacenará el número 13.00.
- También es factible aplicar la fijación de tipo a una variable, por ejemplo obtendremos el mismo resultado, si hacemos:

```
d = (int) f * i ;
```

- En este caso hemos convertido a f en un entero (truncando sus decimales).

Definición de nuevos Tipos (*typedef*)

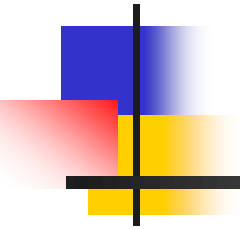
- Permite dar nuevo nombre a tipos de datos que ya existen, siendo estos más acordes con aquello que representan.

  Sintaxis:

  typedef tipo_basico nombre;

- Declaración:
  - typedef float Kg;
  - typedef float Mts;
  - typedef unsigned long double enorme ; // ?? No puede ser unsigned

- Y su uso al declarar variables:
  - Kg peso;
  - Mts longitud;
  - enorme variable1;



3.- Operadores y expresiones

Operadores y expresiones

- **Una expresión** es una sucesión de operadores y operandos debidamente relacionados para formar expresiones matemáticas que especifiquen un cálculo, tal como:

  3 + 6*5.

- Los operadores pueden ser unarios, binarios y ternarios, cuando usan uno, dos o tres operandos.

  - +, - unarios: -4, -5, -6.7
  - +,-,*,/,%, binarios: 4+3, 5*6, 4/2, 8%3
  - ? :, ternario: exp? exp_v : exp_f

Operadores

Operador de asignación (=)

- El operador **=** asigna el valor de la expresión a la variable situada a su izquierda.

- Ejemplos:
  - codigo = 3467;
  - farenheit = 123.456
  - Asociativo por la derecha
    - a = b = c = 45;
    - a = (b = (c = 45));
  - int a,b,c;
    - a = b = c = 5;

Operadores

Operadores de asignación

=	a=b
*=	a*=b
/=	a/=b
%=	a%=b
+=	a+=b
-=	a-=b

Operadores

Operadores aritméticos

Operador	Tipos enteros	Tipos reales
+	Suma	Suma
-	Resta	Resta
*	Producto	Producto
/	División entera: cociente	División punto flotante
%	División entera: resto	

Operadores

Operadores aritméticos

Prioridad (mayor a menor)	Asociatividad
+, - (unarios)	Izquierda a derecha (->)
*, /, %	Izquierda a derecha (->)
+, -	Izquierda a derecha (->)

- Los paréntesis se pueden utilizar para cambiar el orden de evaluación de una expresión determinada por su prioridad y asociatividad.

- Ejemplo:
  - (7 * (10 – 5) % 3) * 4 + 9

Operadores

Operadores de incremento y decremento

Incrementación	Decrementación
++n	--n
n+=1	n-=1
n=n+1	n=n-1

```
n=8
m = ++n;
n = 9;
printf(" n = %d", --n);
printf(" m = %d", m);
printf(" n = %d", n--);
printf(" n = %d", n);

int a = 1, b;
b = a++;
b = ++a;
printf(" a = %d", --a);
printf(" b = %d", b);
```

Operadores

Operadores relacionales

- C no tiene tipos de datos lógicos o booleanos para representar los valores verdadero y falso. En su lugar se utiliza el tipo int para su propósito, con el valor de 0 que representa falso y distinto de cero a verdadero.

Operador	Significado	Ejemplo
==	Igual a	a == b
!=	No igual a	a != b
>	Mayor que	a > b
<	Menor que	a <b
>=	Mayor o igual que	a >= b
<=	Mayor o igual que	a <= b

- Los operadores relacionales tienen menor prioridad que los operadores aritméticos, y asociatividad de izquierda a derecha.

Operadores

Operadores lógicos

Operador	Operación lógica	Ejemplo
Negación (!)	No lógica	!(x>=y)
Y lógica (&&)	Op1 && Op2	m < n && i > j
O lógica (||)	Op1 || Op2	m == 5 || n!=10

- Con los operadores lógicos existen solo dos posibles valores: verdadero y falso.
- El operador ! Tiene prioridad más alta que && y que a su vez tiene prioridad mayor que ||.
- La asociatividad es de izquierda a derecha.
- Evaluación en corto circuito:

(x>5) && (log(x)>=.5)
(n!=0) && (x<1/n)

Operadores

Operadores condicional

- El operador condicional **?:** es un operador ternario que devuelve un resultado cuyo valor depende de la condición comprobada. Tiene asociatividad de derecha a izquierda.

- Formato:

  expresion_c ? expresion_v : expresion_f;

- Ejemplo:

  (valor>0) ? valor++ : valor--;

Bibliografía

- Brian Kernighan; P. J. Plauger (1976), **Software Tools**, Addison-Wesley, pp. 352, ISBN 020103669X.
- Cairó Oswaldo. **Metodología de la programación- Algoritmos, diagramas de flujo y programas-.** Ed. Alfaomega.
- Joyanes Aguilar, Luis. **Fundamentos de programación**. Ed. Mc Graw Hill.
- Joyanes Aguilar, Luis. **Programación en C.** McGraw Hill.
- Schildt, Herbert. **Lenguaje C, Programación Avanzada**. McGraw Hill.
- Bronson, Gary. **Algorithm Development and Program Design Using C**, PWS Publishing Co.; Book and Disk edition (February 15, 1996) ISBN: 0314069879.
- Standish, Thomas. **Data Structures, Algorithms, and Software Principles in C**, Addison-Wesley Pub Co; 1st edition (September 30, 1994), ISBN: 0201591189.
- Linden, Peter. **Expert C Programming**, Prentice Hall, 1994, ISBN 0131774298.
- Reek, Kenneth. **Pointers on C**. Addison Wesley, 1997. ISBN 067399866.
- Kernighan, Brian W. & Dennis, M. Richie. **El lenguaje de programación C**. 2ª Ed. Prentice Hall, 1988.
- http://zinjai.sourceforge.net/