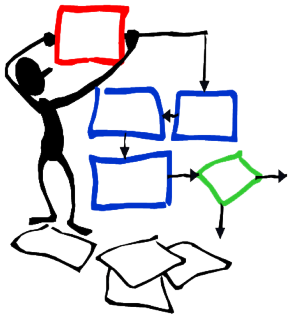
Instituto Politécnico Nacional

Escuela Superior de Cómputo



Algoritmia y programación estructurada

Tema 09: Tipos de datos en C & entrada y salida estándar

M. en C. Edgardo Adrián Franco Martínez
http://www.eafranco.com
edfrancom@ipn.mx
@edfrancom edgardoadrianfrancom

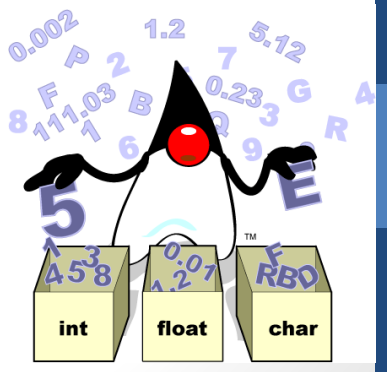
Contenido

- Tipificación en C
- Tipos de datos en C
  - Modificadores de tamaño
  - Modificadores de signo
- Variables
  - Inicialización
  - Captura
  - Salida
- Constantes
- La entrada y salida estándar

Tipificación en C

- El lenguaje C es conocido como un lenguaje **fuertemente tipificado** (***strongly-typed***), esto porque es obligatorio asignar un tipo determinado a cada dato procesado.

- La asignación de tipos tiene dos objetivos principales:
  - Detectar errores de operaciones en programas.
  - Determinar como ejecutar las operaciones



Tipos de datos en C

- Todos los tipos de datos simples o básicos de C son, esencialmente, números:
  - Enteros
  - Números de coma flotante (reales)
  - Caracteres

- C no soporta a un gran número de tipos predefinidos, pero tiene la capacidad para crear sus propios tipos de datos.

Tipos de datos básicos del lenguaje C

Tipo	Tamaño *(Bytes)	Descripción del tipo
char	1 (8 bits)	Carácter o entero de un byte
int	4 (32 bits)	Número entero
float	4 (32 bits)	Números de coma flotante (Reales)
double	8 (64 bits)	Números de coma flotante de doble precisión
void	1 (8 bits)	Tipo nulo (Teóricamente no ocupa memoria)
Punteros	4 (32)	Direcciones de memoria

***Los tamaños de los tipos de datos dependen de la arquitectura (procesador) y el sistema operativo. En este caso se considera una arquitectura x86 & Windows X86.**

- En C los tipos de dato pueden ser modificados en cuanto a su tamaño y capacidad por los **modificadores de tamaño** **short** y **long** (enteros por defecto).
  - short int *(Entero corto de 2 bytes, antes era un int simple)*
  - long int *(Entero largo de 4 bytes actualmente es igual a un entero)*
  - long long *(Entero largo largo de 8 bytes)*
  - long doublé *(Extensión del flotante doublé a 12 bytes)*

- C también incluye **modificadores de signo** **signed** y **unsigned** aplicables a los tipos de dato char, short, int y long,

Capacidades de variables según su tipo

Tipo	Tamaño *(Bytes)	Rango de valores
char = signed char	1	Carácter con signo *(-128 a 127)*
unsigned char	1	Carácter sin signo *(0 a 255)*
int = long = long int=signed int= signed long= signed long int	4	Entero con signo *(-2147483648 a 2147483647)*
unsigned int = unsigned long = unsigned long int	4	Entero sin signo *(0 a 4294967295)*
long long=signed long long	8	Entero largo largo con signo *(-9223372036854775808 a 9223372036854775807)*
unsigned long long	8	Entero largo largo sin signo *(0 a 18446744073709551615)*
float	4	Numero de punto flotante *(1.2 x 10^{-38} a 3.4 x 10^{38})*
double	8	Numero de punto flotante doble *(2.2 x 10^{-308} a 1.8 x 10^{308})*
long double	12	Numero de punto flotante doble *(3.4 x 10^{-4932} a 1.2 x 10^{4932})*

***Los tamaños de los tipos de datos dependen de la arquitectura (procesador). En este caso se considera una arquitectura de 32 bits.**

Variables

- Los datos son almacenados en la memoria de la computadora, en una posición determinada de ella, representada por una dirección de memoria. En esta posición puede escribirse o leerse un dato de acuerdo con el tipo especificado. Sin embargo, debido a la incomodidad de utilizar direcciones para acceder a los datos en memoria se utilizan nombres asociados a estas posiciones. **(Variables).**

- En C, una variable **se declara** indicando el tipo de dato que se alojara y su nombre:

<tipo de dato> <nombre de la variable>;

- Todas las variables de un programa de C deben declararse antes de utilizarse, y es una buena práctica de programación utilizar nombres representativos (en minúsculas) e incluir comentarios sobre su uso.

```
int numero_empleado;          //Número de empleado
float horas;                  //Horas trabajadas
int edad;                     //Edad del empleado
```

Inicialización

- En C, al declarar una variable es posible inicializarla con un valor en una sola sentencia.

<tipo de dato> <nombre de la variable>= <valor>;

```
int base=10,altura=20,area;          //Base, altura y área del terreno
float horas=10.2;                    //Horas de trabajo por metro cuadrado
char empleados=20, categoria='B';//Número de empleados y su categoría
```

Captura

- La función scanf() (scan-format) incluida en stdio.h, representa a una familia de funciones que analizan una entrada de datos con formato y cargan el resultado en los argumentos que se pasan por referencia a dichas funciones:

- **scanf()** lee los datos de entrada en el stdin (flujo de entrada estándar).

- **fscanf()** (file-scanf) lee en un flujo de entrada dado, por lo general un fichero (file) abierto para lectura.

- **sscanf()** (string-scanf) obtiene la entrada que se va a analizar de una cadena de caracteres dada (string).

- Todas las funciones anteriores leen caracteres, los interpretan según un formato, y almacenan los resultados en sus argumentos.

- Cada uno cuenta con varios argumentos: por un lado, un formato de la secuencia del control (se describe más abajo), por otro, un sistema de argumentos del indicador que señala dónde la entrada convertida debe ser almacenada.

- El resultado es indefinido si hay escasos argumentos para dar formato. Si se agota el formato mientras que sigue habiendo las argumentos, los argumentos sobrantes son evaluados pero no procesados de ninguna otra manera.

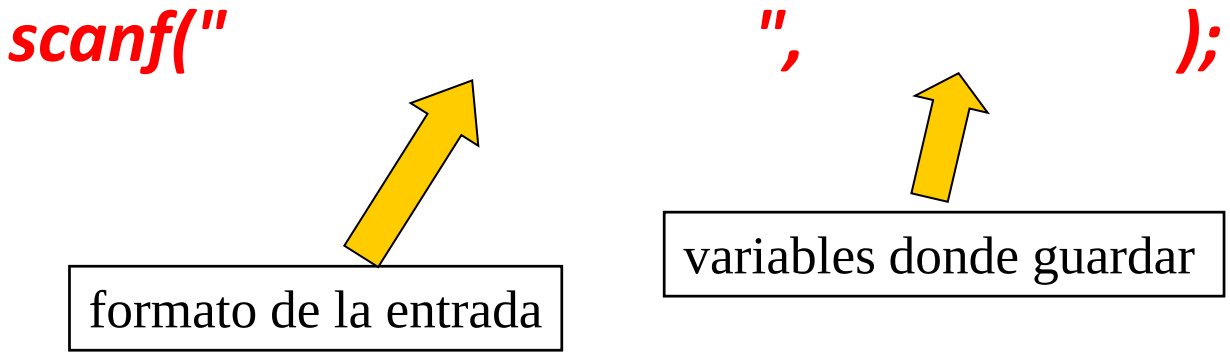
- En el ejemplo siguiente se lee un número entero decimal desde el teclado, que se almacena en la variable **entero1**, posteriormente se leen dos números reales separados por un espacio o salto de línea y se almacenan en las variables **real1** y **real2.**

```
#include<stdio.h>

int main(void)
{
        int entero1;
        float real1,real2;
        scanf("%d",&entero1);
        scanf("%f %f",&real1,&real2);
        return 0;
}
```

Captura (scanf)

- El primer parámetro de **scanf()** es la cadena de **formato**, que indica como debe interpretar el dato leído. El segundo parámetro corresponde a la **variable en memoria** donde quedará almacenado ese dato, y su tipo debe de ser consistente con lo que se espera como entrada.



Captura de caracteres

- La función **scanf()** es útil para capturar por parte de la entrada estándar cualquier tipo de dato. En muchas ocasiones es necesario recibir caracteres del teclado, ya sea para compararlo con su valor real o su valor ASCII. ***Investigación del alumno Codificación ASCII***

- Existen funciones dentro de stdio.h (**getc , getchar y fgetc**), las cuales obtienen el valor de un carácter de la entrada estándar.

```c
#include<stdio.h>

int main(void)
{
		char caracter;
		caracter=getchar();
		printf("El caracter introducido es %c y su valor ASCCI es %d decimal %x hexadecimal",caracter,caracter,caracter);
		return 0;
}
```

Salida

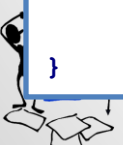
- La función **printf()** y las funciones derivadas **fprintf()**, **sprintf()** y **snprintf()**; permiten escribir una cadena de caracteres en salida estandar (stdout), su nombre proviene de "*print formatted*".

- La instrucción ***printf*** escribe en pantalla
  - Mensajes de texto
  - Valores de variables y expresiones
  - Una mezcla de ambas cosas

  printf ("cadena de formato", arg1, arg2,... argN);

```c
#include<stdio.h>

int main(void)
{
        int entero1=10;
        float real1=0.34;
        printf("El valor del número entero es: %d y del número real es: %f",entero1,real1);
        return 0;
}
```



- La función **printf()** es útil para enviar a la salida estándar cualquier tipo de dato. En muchas ocasiones es necesario mostrar solo un carácter.
- Existen dos funciones dentro de stdio.h (**putchar** y **fputchar**), las cuales muestran el carácter a la salida estándar.

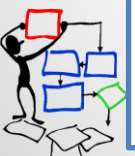
```c
#include<stdio.h>

int main(void)
{
        char caracter, caracter2;
        caracter='A';
        caracter2=65;
        putchar(caracter);
        putchar(caracter2);
        return 0;
}
```

Modificadores de formato

%c Carácter
%d Número entero(int)
%D Número entero long(o también %ld)
%i Número entero(int)
%f Punto flotante(float)
%e Notación científica con e minúscula
%E Notación científica con E mayúscula
%g Utiliza %f o %e según sea más corto
%G Utiliza %f o %E según sea más corto
%o Número octal sin signo
%s Cadena de texto
%u Entero sin signo
%U Entero sin signo long(o también %lu)
%x Hexadecimal sin signo con minúsculas
%X Hexadecimal sin signo con mayúsculas
%p Puntero, dirección de memoria
%n Número de caracteres
%o Formato entero octal
%O Formato entero octal long(o también %lo)
%lf Formato double
%LF Formato long double

Indican el tipo o trato que se le deberá de dar a la variable a la que se hace referencia. Se utilizan dentro de las funciones de entrada y salida; como **printf()** y **scanft().**



Secuencias de escape

- Son un conjunto de caracteres en los textos que son interpretados con algún fin.

Secuencia	Descripción
\n	**Nueva línea**: Coloca el cursor en el principio de la siguiente línea.
\t	**Tabulador horizontal**: Mueve el cursor al siguiente tabulador.
\v	**Tabulador vertical**: Mueve el cursor al siguiente tabulador.
\r	**Retorno de carro**: Coloca el cursor hacia el principio de la línea actual.
\b	**Retroceso**: Backspace o retroceso del cursor.
\\	**Diagonal invertida:** Imprime la diagonal invertida, una sola diagonal sería interpretada como un carácter de escape.
\"	**Comillas**: Imprime la doble comilla. Sin la diagonal invertida, se interpretaría como un inicio o un fin de una cadena.
\a	**Alerta:** Suena la beep del sistema.

El siguiente código muestra el uso de algunas secuencias de escape.

```
#include<stdio.h>
int main(void)
{
	printf("\nHola\tmundo\n¿Como estás? \"Bien\"");
	printf("\aEsto es una\b \\Alerta\\");
	printf("\rNo sobrescribir");
	return 0;
}
```

Constantes

- En programación, una constante es un valor que no puede ser alterado durante la ejecución de un programa.

- Una constante corresponde a una longitud fija de un área reservada en la memoria principal del ordenador, donde el programa almacena valores fijos.

const <tipo de dato> <nombre de la constante> = <valor>;

Constantes

- También es posible declarar una constante utilizando la directiva **define** del preprocesador.

```
#include<stdio.h>
#define PI 3.1416

int main(void)
{
        const float pi=3.1416;
        printf("\nEl valor de PI es: %f",PI);
        printf("\nEl valor de pi es: %f",pi);
        return 0;
}
```

Entrada y salida estándar

- En lenguaje C no existen palabras reservadas para realizar entradas y salidas. Para ello, el programador puede hacer uso de las funciones de entrada y salida estándar proporcionadas por la biblioteca estándar de lenguaje C, como son printf y scanf, entre otras que se encuentran declaradas en la cabecera **<stdio.h>.**

- **stdio.h** *"standard input-output header"* **(cabecera estándar E/S)**, es la biblioteca estándar del lenguaje de programación C, el archivo de cabecera que contiene las definiciones de macros, las constantes, las declaraciones de funciones y la definición de tipos usados por varias operaciones estándar de entrada y salida.

Funciones de <stdio.h>

fclose	fopen	freopen	fdopen
remove	tmpfileme	rewind	auto
clearerr	feof	ferror	fflush
fgetpos	fgetc	fputsputcgets	ftell
fseek	fsetpos	fread	fwrite
getc	**getchar**	gets	**printf**
fprintf	sprintf	snprintf	vprintf
perror	putc	**putchar**	fputchar
scanf	fscanf	sscanf	vfscanf
vscanf	vsscanf	setbuf	setvbuf
tmpnam	ungetc	puts	

Re-direccionar la entrada y salida estándar

- La **entrada estándar** es el lugar desde donde un comando obtiene la información necesaria de entrada. Habitualmente, por omisión, ésta es el teclado.

- La **salida estándar** es el lugar a donde un comando envía el resultado de su ejecución, por omisión, este lugar es la pantalla).

  - < Acepta la entrada estándar desde un archivo.

  - > Envía la salida estándar a un archivo.

  - >> Añade la salida estándar a un archivo.

  - | Conecta la salida estándar de un comando con la entrada estándar de otro.