

### Operadores

/	Or
&&/&	And
==	Igual que
!	Not
!=	Diferente de
+ - /	Operadores aritméticos
* %	
++/ -	Incremento/Decremento
-	
< <=	Comparaciones
> >=	
?:	condición? Expresión 1: Expresión 2; Si se cumple la condición se ejecuta la expresión 1 y si no se cumple se ejecuta la expresión 2

### Comentarios y salto de línea

//	Comentario de una línea
/* */	Comentario en bloque
\n	Salto de línea

### Tipos de datos

char	'a'	Rango: 0..255	%c
short	-15	Rango:-128...127	%hi
int	1012	Rango:-32768...32767	%i
unsigned int	128	Rango:0...65535	%u
long	123456		%li
float	15.4		%f
double	123123.123123		%lf
long double	Número muy grande con decimales		%Lf
char X[50]	string		%s

En el string, "X" es la variable y [50] es la cantidad de espacios

### Directivas del preprocesador

# include: Para la inclusión de archivos  
 # define: Para la definición de macros  
 # if: Se usa para inclusiones condicionales  
 # undef **numero**: Indefino a **numero**, es decir, anulo su definición previa

### Estructura general de un programa

```
#include <stdio.h> /* Cabecera estándar de entrada/salida de datos/*
Prototipos /* Aquí se definen los prototipos de las funciones se usarán dentro de la función main /*
int main(){ // Función principal
instrucciones;
return 0; // Buena praxis de programación
}
```

### Entradas y salidas de datos

printf( ) Muestra en pantalla los datos

scanf(" %i",&variable) Guarda un dato que el usuario digitó

gets( ) Se usa para cadenas de strings, dado que el scanf solo lee hasta que haya un espacio

puts( ) Muestra datos en pantalla pero solo si está dentro de un condicional

%i: Define el tipo de dato que se está guardando.  
 &variable: Define en qué posición de memoria se guardará el dato, seguido del nombre de la variable donde se guardará.

### Sentencia if-else

```
if(condición){
instrucción;
}
else{
instrucción;
}
```

### Sentencia switch

```
switch(selector){
case etiqueta1: sentencia1;break;
case etiqueta2: sentencia2;break;
case etiqueta3: sentencia3;break;
default: sentencia;
}
```

-El selector de un switch solo puede ser de tipo entero o char.  
 -Si no se cumple ninguno de los casos, entra al default.

### Funciones matemáticas

ceil(x): Redondea a techo siempre  
 fabs(x): Devuelve el valor absoluto de x  
 floor(x): Redondea a piso  
 fmod(x,y): Calcula el residuo de la división de x/y  
 pow(x,p): Eleva x a la potencia p  
 sqrt(x): Devuelve la raíz cuadrada de x

### Otros

getch( ): Para hacer que el ejecutable no se cierre inmediatamente, si no hasta que se presione una tecla  
 fflush(stdin): Para limpiar el buffer

Para usar **getch( )** ocupó incluir la biblioteca <conio.h>.

### Bucles

#### While

```
while( ){
instrucciones;
}
```

#### For

```
for(inicialización;condición;incremento){
instrucciones;
}
```

#### Do while

```
do{
instrucciones;
}while( );
```



### Arrays unidimensionales

Sintaxis del array:

```
TipoDeDato nombreArray [numeroElementos]
```

Ejemplo:

```
int numeros[5] = {1,2,3,4,5}
```

-{1,2,3,4,5}: Forma de asignar elementos a un array o también se le pueden pedir al usuario.

```
-int numeros[0]=1
```

### Arrays Bidimensionales (matrices)

Sintaxis para la declaración:

```
TipoDeDato nombreDelArray [numeroFilas]
```

```
[numeroColumnas]
```

Ejemplo:

```
int matriz[2][3] = {{41,42,43},{44,45,46}};
```

-{{41,42,43},{44,45,46}}: Así declaramos los elementos que van a rellenar la matriz, el primer grupo de números es la primera fila, por lo tanto el segundo es la segunda fila y se agrupan de 3 en 3 porque es el número de columnas.

```
- int matriz[0][1]=42
```

### Estructuras

Es una colección de datos de diferente tipo.

Declaración de una estructura:

```
struct nombreDeLaEstructura {
```

```
tipoDeDato nombreDeMiembro;
```

```
tipoDeDato nombreDeMiembro;
```

```
tipoDeDato nombreDeMiembro;
```

```
}; Aquí declaro las variables, las cuales
```

contienen cada uno de los miembros de la estructura y puedo acceder a ellos por medio del .

Ejemplo:

```
struct persona{
```

```
char nombre[20];
```

```
int edad;
```

```
}
```

```
persona1= {"Briana",17},
```

```
persona2= {"Paula",19};
```

```
int main(){
```

```
printf("Su nombre es: %s ",persona1.nombre);
```

```
printf("\n Su edad es: %i ", persona1.edad);
```

```
return 0;
```

### Estructuras (cont)

```
}
```

También le puedo pedir al usuario que me

digite la información de las variables. Por

ejemplo la edad la guardaría con un

```
scanf("%i",&persona1.edad);
```

### Arreglos de estructuras

Ejemplo:

```
struct persona{
```

```
char nombre[20];
```

```
int edad;
```

```
}personas[5]; // persona1,...,persona5
```

```
int main() {
```

```
int i;
```

```
for(i=0;i<5;i++) {
```

```
fflush(stdin);
```

```
printf("%i. Digite su nombre: ",i+1);
```

```
gets(personas[i].nombre);
```

```
printf("%i. Digite su edad: ",i+1);
```

```
scanf("%i",&personas[i].edad);
```

```
}
```

### Punteros

Declaración de un puntero:

```
tipoDeDato *nombreDePuntero;
```

Ejemplo:

```
int main ( ){
```

```
int numero=50;
```

```
int *p_numero; //puntero
```

```
p_numero = &numero;
```

```
// &numero: Posición de memoria de numero
```

```
printf("Dato: %i",numero); //imprime 50
```

```
printf("Dato: %i",*p_numero); /*Así también me
```

```
imprime 50 que es el valor de la variable
```

```
numero. */
```

```
printf("Posicion de memoria: %p ",p_numero);
```

```
printf("Posicion de memoria: %p ",&numero);
```

```
/*Las dos formas imprimen la posición de
```

```
memoria*/
```

```
return 0;
```

```
}
```