Escuela Politécnica Superior

SEMINARIO-TALLER DE SOFTWARE (STI-S)

UNIDAD 4. Programación en C

Sesión 3

Índice

Índice

Arrays simples y bucles

- Los arrays se suelen manipular mediante ***bucles***.

```
void main() {
      int  v[20], i ;
      for (i=0; i<20; i++){
        scanf("% d", &v[i]);
      }
      for(i=0; i<20; i++){
        printf("%d", v[i]);
      }
}
```

Arrays Multidimensionales y bucles

- Los arrays multidimensionales se suelen manipular mediante ***bucles anidados***:

```
void main(){
    int m[10][20] ;
    int i,j;
    for (i=0; i<10; i++){
      for ( j=0; j<20; j++){
          scanf ("%d", &m[i][j]);
      }
     }
}
```

Índice

Cadenas de Caracteres

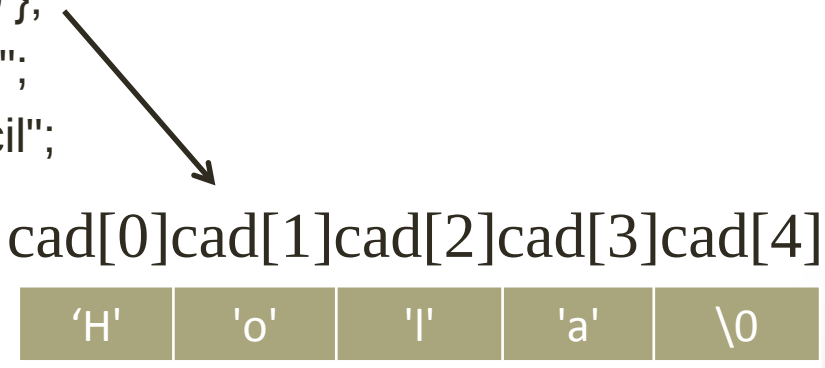
- Las cadenas de caracteres se manejan como ***arrays*** de ***char***.
- Fin de cadena se marca con el carácter **'\0'**.
- Su tamaño será su longitud + 1.

- Definición:

  char nombre[tamaño];

- Ejemplos:

```
char cad[5] = {'H', 'o', 'l', 'a', '\0'};
char cad2[20] = "Hola cadena";
char cad3[ ] = "Así es más fácil";
```



Cadenas de Caracteres

- Funciones más comunes para trabajar con cadenas de caracteres:

Funciones para cadenas de caracteres	
char* strcpy(char *cad1, char *cad2)	Copia cad2 en cad1
int strcmp(char *cad1, char *cad2)	Compara cad2 y cad1. Devuelve un número entero mayor, igual, o menor que cero, según la **s1** es mayor, igual, o menor que **s2**
int strlen(char *cad)	Devuelve la longitud de **cad**
char *strtok(char *s1, const char *s2);	Rompe la cadena **s1** en segmentos o tókens dependiendo del delimitador **s2**

Ejemplo: int verificar () {

```
        char secreto="abracadabra";
        char password[50] ;
        scanf("%s", password);
        i f (strcmp(secreto,password) == 0)  return 1 ;
        else return 0;
    }
```

Índice

Enumerados

- Los enumerados se utilizan para definir los posibles valores que las variables pueden tomar.
- Por ejemplo: se puede crear un enumerado para los días de la semana.

```c
#include <stdio.h>

typedef enum {
  lunes,
  martes,
  miercoles,
  jueves,
  viernes,
  sabado,
  domingo
}dias;

void main(){
  dias dia1 = lunes;
  dias dia2 = martes;
  printf("%d\n", dia1);
  printf("%d\n", dia2);
}
```

Índice

Castings

- Un casting (casteo) es una manera de convertir una variable de un tipo a otro.
- La manera de hacer *castings* es la siguiente:

(tipo de dato) expresión

```c
#include <stdio.h>
void main(){
    int a, b;
    a = 7;
    b= 2;
    float c = a/b;
    printf("%f\n", c);
}
```

- ¡¡ El resultado es 3 !!

```c
#include <stdio.h>
void main(){
    int a, b;
    a = 7;
    b= 2;
    float c = (float)a/b;
    printf("%f\n", c);
}
```

- El resultado es 3.5

Castings

- Cualquier carácter puede ser representado como un integer:

```c
#include <stdio.h>
void main(){
   char myChar = 'a';
   printf("Char value: %c\n", myChar);
   printf("Int. value: %d\n", myChar);
}
```

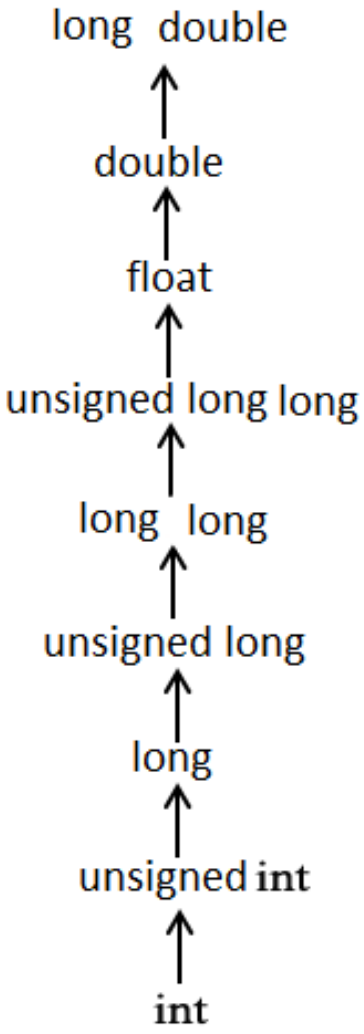
```
agpardo@virtualBox:~/Escritorio$ ./prueba
Char value: a
Int. value: 97
```

- Para saber el valor decimal de cualquier carácter solo hay que mirar la tabla ASCII:

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
64	40	100	@	@	96	60	140	`	`
65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d

Castings

- En las operaciones aritméticas, cuando los tipos de los operadores son diferentes, se realizan castings hasta que ambos tipos son los mismos.
- La manera en la que se realizan los castings está representada en la siguiente jerarquía.



Índice

Constantes

- Las constantes son variables cuyos valores no cambian durante la ejecución.
- Se pueden crear constantes de *integer*, *float*, *double, carácter* o de cadenas de caracteres.
- Se pueden crear constantes de dos maneras diferentes: usando *#define* o *const:*

```
#include <stdio.h>

#define MAX_USERS 100

void main(){
   const char* PASS_TAG="pass";
   printf("The max. number of users is: %d\n", MAX_USERS);
   printf("The pass. tag is: %s\n", PASS_TAG);
}
```

Índice

Funciones

- Las funciones son trozos de código reutilizables
  - Implementan una tarea concreta.
  - Pueden devolver un valor (la instrucción ***return*** indica qué valor devolver y termina la ejecución de la función).
  - Si no se devuelve nada el tipo de retorno es ***void*****.**
  - Admiten argumentos de entrada.
  - **main()** es una función especial.
- Deben ser definidas antes que utilizadas
- Regla: *Meter en una función el código que se use más de una vez*

- Definición:

  tipo_devuelto nombre_función (parámetros) {cuerpo}

Funciones

- Ejemplo:

```
int sumar (int a, int b) {
  int c = a + b;
  return c;
}
int main ( ) {
  int var1=3 , var2=2, resultado;
  resultado = sumar (var1, var2) ;
  printf("Resultado = %d", resultado);
  return 0 ;
}
```

Índice

Ámbito de las variables

- Las variables existen dentro del bloque en que se definieron
  - Una variable definida en una función sólo existe en dicha función.
  - Dos variables con el mismo nombre en dos funciones son variables distintas

- **Variables locales**: Variables definidas dentro de una función.

```
int main ( ) {
  int a=2, b=4, c=0;
  printf("% d" , sumar(a,b));
  printf("%d",c);
  return 0 ;
}
```

Ámbito de las variables

- **Variables globales:** Se definen fuera de todo el bloque de código y se pueden utilizar en todas las funciones.

```
int a, b ,c;
void sumar () {
  int c = a+b;
  printf ("% d", c);
}
int main ( ) {
  a = 2;
  b = 4;
  sumar();
  return 0 ;
}
```

- Cuidado con mezclar variables locales y globales con el mismo nombre.

Índice

Paso de parámetros

- **Por valor:** los argumentos que se pasan a una función son copias locales.
  - Se crea una variable local.
  - Se copia el valor del argumento en esa variable.

```
int sumar (int a, int b) {
  int c = a + b;
  return c;
}
int main ( ) {
  int var1=3 , var2=2, resultado;
  resultado = sumar (var1, var2) ;
  printf("Resultado = %d", resultado);
  return 0 ;
}
```

Paso de parámetros

- **Por referencia:** Envía como argumento la dirección de memoria (puntero) en la que se guarda la variable.
  - Se antepone un & antes de la variable al invocar la función.
  - Hay que poner un * antes de la variable en la cabecera de la función.
- Cualquier cambio a la variable pasada por referencia se propaga

```
void sumar ( int a , int b , int *c ) {
    *c = a+b;
}
void main( ) {
   int valor1=2, valor 2=3, c=0;
   sumar ( valor1 , valor2 , &c);
   printf ("% d" , c);
}
```

Paso de parámetros

- **Parámetros de la función *main*:** Los argumentos escritos por el usuario en la línea de comandos.
  - ***Primero*** (argc): de tipo int. Número de argumentos introducidos.
  - ***Segundo*** (argv): array de cadenas (char *). Indica el contenido de esos argumentos.

```
int main (int argc , char *argv []) {
    int i ;
    for (i=0; i<argc ; i++){
            printf ( "% s \n", argv[i]);
    }
}
```

Devolución del programa

- **Salida de la función *main*:** Cuando la función main concluye con un return, el valor devuelto por return puede ser consultado en la terminal con el comando:
  - **echo $?**
- Ejemplo:

```
int main (int argc , char *argv []) {
        if(argc < 2)
                return -1;
        //Código
        return 0;
}
```

- > echo “La salida del programa es $?”