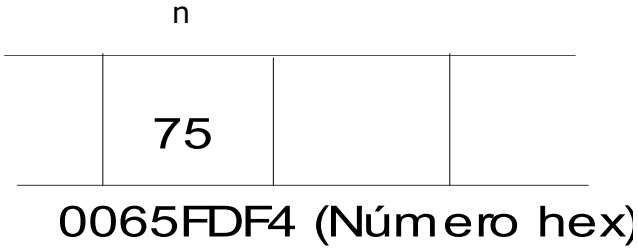
CAPÍTULO 5

PUNTEROS

5.1 Direcciones de memoria

Cada vez que se declara una variable, el compilador establece un área de memoria para almacenar su contenido, que se guarda en una posición específica de la memoria identificada por su DIRECCIÓN. Para obtener la dirección de una variable en C se utiliza el OPERADOR DE DIRECCIÓN **&**.

Por ejemplo, si n es una variable entera &n devuelve un número hexadecimal (en base 16) que corresponde a la dirección de n en la memoria RAM. En el siguiente gráfico se tiene un ejemplo, en el que la variable n contiene el valor 75 y cuya dirección en memoria es 0065FDF4.



Entonces, el siguiente programa:

```
#include<stdio.h>
void main(   )
{
int n=75;
printf("n=%d \n",n);
printf("&n=%p \n",&n);
printf("&n=%X \n",&n);
}
```

imprime el resultado:

```
n=75
&n=0065FDF4
&n=65FDF4
```

En el ejemplo anterior se ha utilizado un especificador de formato que todavía no conocíamos: %p. Se emplea para mostrar una dirección de memoria. Imprime el valor en hexadecimal. Se puede considerar equivalente a %x.

5.2 Punteros

Un puntero es una dirección de memoria (dada por un número hexadecimal). Una variable puntero, contiene una dirección de memoria.

5.2.1 Declaración de punteros

La sintaxis es la indicada a continuación:

Tipo de Dato * identificador

Véanse los siguientes ejemplos:

```
int  *ptr1;      /*Variable puntero a un tipo de dato int*/
char  *ptr2;     /* Variable puntero a un tipo de dato char*/
float  *ptr3;    /*Variable puntero a un tipo de dato float*/
```

5.2.2 Inicialización estática de punteros

El método de inicialización estática consiste en definir una variable y un puntero del mismo tipo que la variable y, a continuación, asignar al puntero la dirección de la variable utilizando el operador de dirección &.

Ejemplo:

```
int  i;
int *p;
 p=&i;
```

5.2.3 Indirección de punteros

El uso de un puntero para obtener el valor al que apunta, es decir, el dato almacenado en la posición de memoria señalada por él, se denomina INDIRECCIONAR el puntero. Para ello se utiliza el operador de INDIRECCIÓN *****.

Por ejemplo,

```
int e=5;        /*Declaración e inicialización de la variable e*/
int *p;         /*Declaración de la variable puntero p*/
p= &e ;         /*  Inicialización del puntero*/
printf("%d \n",*p);   /* Imprime el valor de *p que es 5*/
```

Si se escribe en un programa,

```
float *px;
*px=23.5;
```

se generaría un error, pues px no tiene dirección asignada.

Véanse a continuación algunas asignaciones simples entre punteros:

Sean dos punteros p1 y p2,

p1=p2; /*p1 apuntará a la misma variable que apunte p2*/

*p1=*p2; /* la variable apuntada por p1 tomará el valor que tenga la variable apuntada por p2*/

Para afianzar los conceptos de punteros estudiados hasta ahora se propone la realización del siguiente ejercicio: determinar la salida a pantalla del programa escrito a continuación.

```
#include<stdio.h>
void main( )
{
 int a, b, c, *p1, *p2;
 p1 = &a;
*p1 = 1;
 p2 = &b;
*p2 = 2;
 p1 = p2;
*p1 = 0;
 p2 = &c;
 *p2 = 3;
 printf("%d %d %d\n", a, b, c);
  p1=p2;
  p2=&a;
  *p2=*p1;
  printf("%d %d %d\n", a, b, c);
 }
```

La salida a pantalla es:

1 0 3
3 0 3

5.3 El Puntero NULL

El puntero NULL es una dirección de memoria no válida. Se utiliza para proporcionar al programa un medio para saber cuando una variable puntero no direcciona a un valor válido o no apunta a ninguna dirección de memoria. Para declarar un puntero nulo se utiliza la sintaxis:

Tipo de dato * nombre del puntero = NULL

Por ejemplo,

```
char *p = NULL;
```

Se empleará más adelante en este capítulo.

5.4 Aritmética de punteros

Los punteros pueden ser utilizados como cualquier otra variable, sin embargo, se deben considerar algunas restricciones y reglas:

1. Además de los operadores * y & sólo hay cuatro operadores que pueden ser aplicados a las variables puntero: +, ++, - y --.

2. Sólo se pueden sumar o restar cantidades enteras.

3. La aritmética de punteros difiere de la ordinaria en el hecho de que se realiza relativa al tipo de cada puntero. Es decir, cada vez que un puntero es incrementado, apunta al siguiente ítem, de acuerdo a su tipo, del que está apuntando en el momento actual. Por ejemplo, si se define `int * p` y `p` contiene la dirección 200, como el tipo base de los enteros tiene 4 bytes, `p+1` contendrá la dirección 204. (En este ejemplo, se han utilizado números decimales en lugar de hexadecimales para las direcciones). Sólo en el caso de los caracteres, la aritmética de los punteros coincide con la ordinaria.

A continuación aparecen algunas operaciones compuestas aplicadas a punteros:

```
a =*p++        equivale  a    a=*p;
                              p++; (p=p+1)

a =*++p        equivale  a    p++; (p=p+1)
                               a=*p;

a =++*p        equivale  a    *p +=1; (*p=*p+1)
                               a=*p;

a = (*p)++     equivale  a     a=*p;
                               (*p)++;  (*p=*p+1)

*p++ =*q--     equivale  a     *p=*q;
                                q=q-1;
                                p=p+1;

(*p++) += i    equivale  a     *p=*p+i;
                               p++; (p=p+1)
```

5.5 Punteros y vectores (Tablas unidimensionales)

El nombre de una tabla es simplemente una constante puntero que apunta al primer elemento de la tabla.

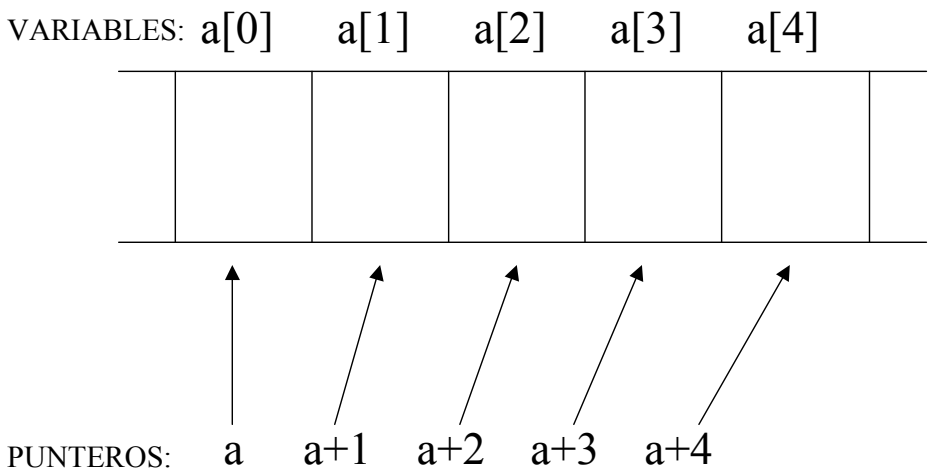
Sea la siguiente declaración e inicialización de un vector:

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

se tiene que,

`a` es una constante puntero que apunta a `a[0]`.
`a+1` apunta a `a[1]`, y así sucesivamente.
`*a` es equivalente al valor de `a[0]`.
`a` es equivalente a `&a[0]`.

El siguiente gráfico clarificador la explicación:



A continuación se escribe un sencillo ejemplo que ilustra la forma de manejar un vector (lectura y escritura) por el método tradicional y mediante la utilización de punteros:

1) Método tradicional

```
#include <stdio.h>
void main ( )
{
int i;
float v[5];
for(i=0;i<5;i++)
{
  printf("Introduce la componente %d: ",i+1);
  scanf("%f", &v[i]);
}
printf("Las componentes del vector son:\n");
for(i=0;i<5;i++) printf("%f ",v[i]);
}
```

2) Utilizando la constante puntero

```
#include <stdio.h>
void main ( )
{
int i;
float v[5];
for(i=0;i<5;i++)
{
printf("Introduce la componente %d: ",i+1);
scanf("%f", v+i);
}

printf("Las componentes del vector son:\n");
for(i=0;i<5;i++) printf("%f ",*(v+i));
}
```

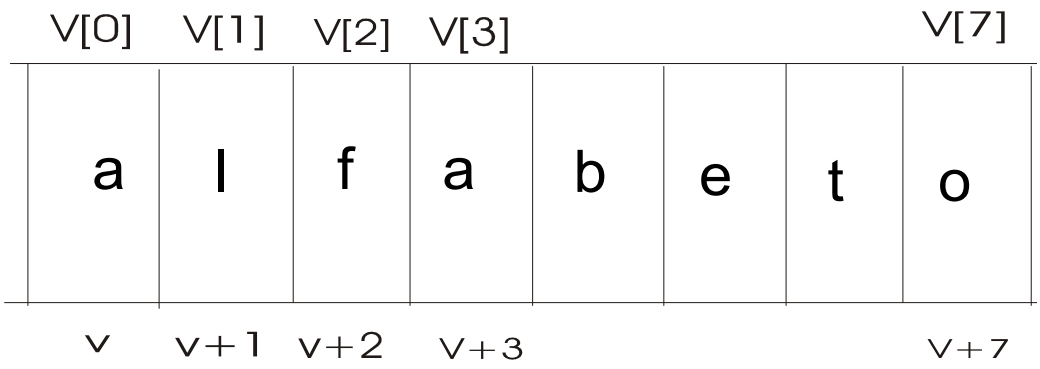
3) Utilizando una variable puntero

```
#include <stdio.h>
void main ( )
{
int i;
float v[5], *p;
p=v;
for(i=0;i<5;i++)
{
printf("Introduce la componente %d: ",i+1);
scanf("%f", p++);
}
p=v;
printf("Las componentes del vector son:\n");
}
```

Otro ejemplo para cadenas:

```
char V[9] = "alfabeto";
```

La ubicación en memoria de la cadena v se ilustra en el siguiente gráfico:



en donde v+i son constantes de tipo puntero.

Ejemplo de programa y salida a pantalla:

```
#include <stdio.h>
void main( )
{
int i;
char *p, v[9]="alfabeto";
p=v;
for(i=0;i<8;i++)
 {
  printf("%c \n",*p);
  p=p+1;
 }
}
```

sale *por pantalla:*

```
a
l
f
a
b
e
t
o
```

Cuando se utilizan constantes de cadena, se puede decir que los dos fragmentos de programa siguientes son equivalentes:

Fragmento de programa 1:

```
char cad[ ]="Una cadena";
char *p;
p=cad;
printf(cad); /*sería equivalente a printf(p);*/
```

Fragmento de programa 2:

```
/*dado que p y cad son equivalentes, se podría evitar la definición de cad*/
char*p="Una cadena"
printf(p);
```

La utilización de punteros a cadenas con esta segunda notación resultará especialmente útil cuando se tengan que manejar varias cadenas, como se comprobará más adelante.

Como ejercicio, se pide al lector que diseñe un programa C utilizando punteros, que lea una cadena por teclado y devuelva la cadena del revés.

Solución:

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

void main( )
{
char cadena1[80],cadena2[80]="";
char *p1,*p2;
int i;
printf("Introduzca una cadena menor de 80\n");
gets(cadena1);
p1=cadena1;
p2=cadena2;
p1=p1+strlen(cadena1)-1;
```

```
for(i=1;i<=strlen(cadena1);i++)
{
     *p2=*p1;
      p1--;
          p2++;
 }
 printf("La cadena al revés es: %s",cadena2);
 }
```

5.6 Tablas de Punteros

En C se pueden definir tablas de punteros. La siguiente sentencia de declaración,

```
int *a[10], v;
```

define un vector de diez elementos cada uno de los cuales es una variable puntero a una variable entera, por ejemplo, se podría hacer la siguiente asignación:

```
a[0]=&v;
```

Una utilidad directa de las tablas de punteros es el manejo de varias cadenas de caracteres. Tal como se estudió en el apartado anterior se puede definir una cadena de la forma:

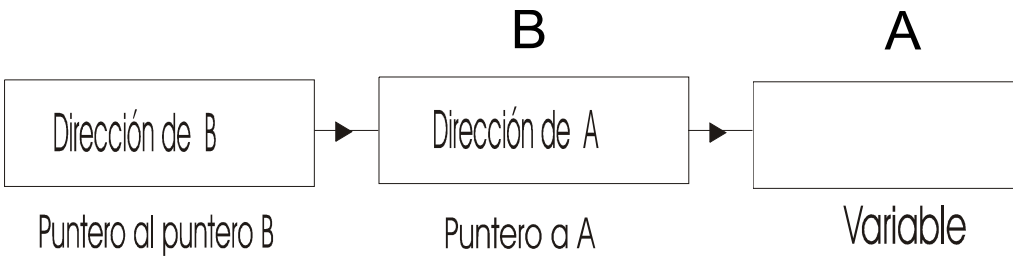
```
char *p = "Una cadena";
```

Con esta nomenclatura directa y usando tablas de punteros se pueden crear punteros que apunten a varias cadenas:

```
char*p[ ]={"Una cadena","Otra cadena"}
```

5.7 Punteros de punteros (Indirección múltiple)

En C es posible tener un puntero que apunta a otro puntero:



El operador INDIRECCIÓN MÚLTIPLE es **.

Con la siguiente sentencia

```
char  **mp;
```

se declara mp como un puntero a un puntero de tipo carácter.

Sea el fragmento de programa:

```
char  **mp,  *p, ch;
p=&ch;              /*dirección de ch*/
mp=&p;              /* dirección de p*/
**mp='A';   /*Asigna a ch el valor A usando la  indirección múltiple*/
```

su explicación puede ser la siguiente: imaginemos que ch se guarda en una dirección de memoria 200, p entonces guarda el valor 200, y como es una variable, se almacena en otra dirección de memoria, por ejemplo la 450; como mp apunta a p, guarda la dirección de p, es decir el valor 450. Con estos datos, las siguientes expresiones arrojarían el resultado que se indica:

```
mp → 450
*mp → 200
p → 200
*p → 'A'
**mp → 'A'
```

Con datos reales, continuando el código del programa anterior:

```
printf("%p\n",mp);          →0012FF78
printf("%p\n",p);           →0012FF74
printf("%p\n",*mp);         →0012FF74
printf("%c\n",*p);          → A
printf("%c\n",**mp);        → A
```

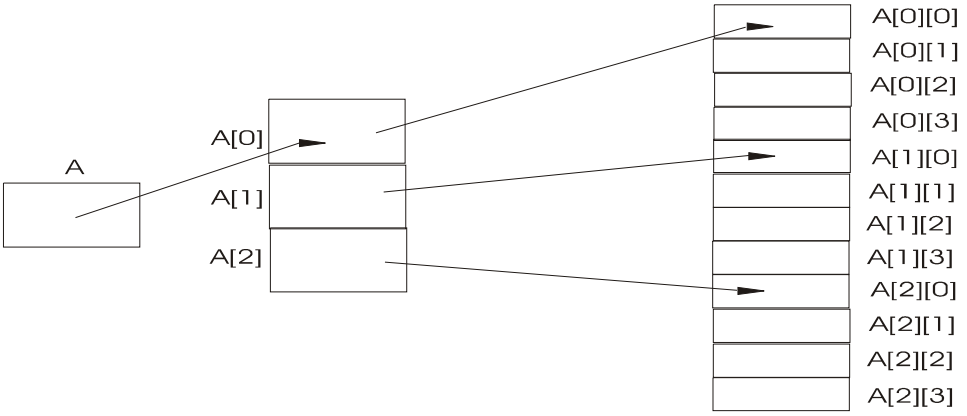
5.8 Tablas multidimensionales y punteros

Para analizar cómo están relacionadas las tablas multidimensionales con los punteros, consideremos el siguiente ejemplo, referido a una matriz:

Se declara la matriz A entera de 3 filas y 4 columnas:

```
int A[3][4];
```

El conjunto de punteros y variables que se generan en memoria aparecen en el gráfico siguiente:



Se tiene que:

1. A es un puntero constante que contiene la dirección del vector de punteros A[ ], cada uno de los cuales contiene la dirección del primer elemento de cada fila de la matriz. Es decir, A[i] contiene la dirección de A[i][0].

2. De lo dicho, se tiene que para poder acceder a un elemento de la matriz se pueden utilizar tres técnicas:

    a) La notación tradicional: **A[i][j]**.
    b) El uso del vector de punteros A[ ]: ***(A[i]+j)**.
    c) El uso del nombre de la matriz A: ***(*(A+i)+j)**.

Estos conceptos se ilustran en los siguientes ejemplos:

El programa:

```
#include <stdio.h>
 void main( )
{
 int a[3][4]={1,2,3,4,10,20,30,40,100,200,300,400};
 int *p0, *p1, *p2;
 p0=a[0] ; p1=a[1] ; p2=a[2];
 printf("p0=%p p1=%p p2=%p \n",p0,p1,p2);
}
```

imprime el resultado:

P0=0065FDC8 p1=0065FDD8 p2=0065FDE8

Si las sentencias 6, 7 y 8 se cambian por p0=&a[0], p1=&a[1] y p2=&a[2], la compilación nos daría un ERROR puesto que a[0],a[1] y a[2] son constantes de tipo puntero y no variables.

Véase otro ejemplo. Sea el programa:

```
#include <stdio.h>
 void main( )
{
 int j, a[3][4]={1,2,3,4,10,20,30,40,100,200,300,400};
/* Escritura de los elementos de la  3ª fila  */
 for(j=0;j<4; j++)
 printf("a[2][%d]=%d \n",j, a[2][j]);
 for(j=0;j<4; j++)
 printf("a[2][%d]=%d \n",j, *(a[2]+j));
for(j=0;j<4; j++)
printf("a[2][%d]=%d \n", j, *(*(a+2)+j));
}
```

La salida del programa es:

a[2][0]=100
a[2][1]=200
a[2][2]=300
a[2][3]=400 (repetido 3 veces)

5.9 Dimensionamiento dinámico

Dimensionamiento dinámico es el proceso por el cual, la memoria es localizada en tiempo de ejecución del programa. Este procedimiento se utiliza para:

- Optimizar el uso de la memoria.
- Crear estructuras de datos dinámicas como, por ejemplo, las listas y los árboles.

El proceso de creación de una variable dinámica consiste en localizar una zona de memoria libre del tamaño adecuado y hacer que una variable puntero apunte a ella. A esa zona de memoria, posteriormente se asignarán valores del mismo tipo que el del puntero creado.

Hasta ahora, cuando en un programa se utilizaba una tabla y su tamaño no era conocido (variable), la única posibilidad que existía era mayorar el tamaño de la tabla para luego utilizar sólo una parte, con el gasto de memoria que implicaba (véase el ejercicio 18 de las prácticas del capítulo 4). Con la técnica de dimensionamiento dinámico se puede ajustar en memoria el tamaño real que tenga la tabla en cada ejecución del programa

5.9.1 Herramientas para el dimensionamiento dinámico

- **El operador sizeof (stdio.h)**

El operador sizeof da como resultado el tamaño en bytes de su operando. El operando puede ser el *identificador* de un objeto de datos o el *tipo* del objeto de datos. Por ejemplo, el programa:

```
#include <stdio.h>
void main( )
{
 printf("El tamaño de un entero es %d \n", sizeof(int));
 printf("El tamaño de un float es  %d  \n", sizeof(float));
 printf("El tamaño de un double es  %d  \n",sizeof(double));
 }
```

imprime como resultado:

El tamaño de un entero es 4
El tamaño de un float es 4
El tamaño de un double es 8

El tamaño ocupado por un puntero se calcula:

sizeof (int*) /*tamaño que ocupa en memoria un puntero a entero → 4 BYTES)
sizeof (char*) /*tamaño que ocupa en memoria un puntero a carácter → 4 BYTES)

El lector debe recordar que un puntero siempre memoriza una dirección de memoria, es decir, un entero y por ello siempre va a ocupar 4 bytes.

- **La función malloc (stdlib.h)**

La sintaxis de esta función es:

void *malloc(size-t número de bytes)

Esta función busca una zona libre de la memoria en la que puede localizar la cantidad de memoria que indica su argumento. Si puede realizar esta operación sin problemas retorna un puntero al comienzo del bloque de memoria localizado. Si hay algún problema, retorna un puntero NULL.
Para el buen funcionamiento de los programas, es conveniente confirmar que **malloc ( )** tiene éxito antes de utilizar el puntero que retorna. Esto se puede realizar del siguiente modo:

```
p=malloc(size);
if(!p) /*o también if(p==NULL)*/
{
 printf ("Error de localización");
 exit(1);
}
```

- **La función free( ) (stdlib.h)**

Para liberar memoria se utiliza la función free con un puntero al comienzo del bloque de memoria previamente localizada:

```
free(p);
```

5.9.2 Ejemplo de dimensionamiento dinámico de un vector

Véase un ejemplo de cómo dimensionar un vector cuyos elementos son números reales en doble precisión:

```
#include<stdio.h>
#include<stdlib.h>
void main ( )
{
double *v;
int n,i;
/* La dimensión del vector se lee por teclado */

printf("Introduce el número de elementos del vector:");
scanf("%d",&n);

/* Dimensionamiento Dinámico */
v=(double *)malloc( n*sizeof(double)) ;  /* (double *) es optativo */

/* Ahora se comprueba si el dimensionamiento ha sido posible */
if (!v)
{
printf("Error de localización");
exit(1);
}
/*  Inicialización del vector e impresión */

for(i=0;i<n;i++)
{
v[i]=i;
printf("v[%d]=%d \n",i,v[i]);
}
free(v);
}
```

5.9.3 Ejemplo de dimensionamiento dinámico de una matriz

El siguiente programa dimensiona dinámicamente una matriz de elementos enteros:

```
#include<stdio.h>
#include<stdlib.h>
void main( )
{
int **A;
int i,j,F,C;
printf("Introduce el número de filas de la matriz");
scanf("%d",&F);
printf("Introduce el número de columnas de la matriz");
scanf("%d",&C);

/* Asignación de memoria para el vector de punteros */

A=(int **) malloc(F*sizeof(int *));

/* Comprobación de la asignación  */
if (A= = NULL)
{
printf("Espacio insuficiente de memoria");
exit(0);
}
/* Asignación de memoria para cada una de las filas de la matriz y comprobación de si la asignación ha sido correcta */
for(i=0;i<F;i++)
{
A[i]=(int *) malloc(C*sizeof(int));

if (A[i]= = NULL)
{
printf("Espacio insuficiente de memoria");
exit(0);
}
}
```

```
   /* Inicialización de la matriz A a un valor y comprobación del resultado */

     for (i=0;i<F;i++)
   {
   for (j=0;j<C;j++)
   {
   A[i][j]=i+j;
   printf("A[%d][%d]=%d \n",i,j,A[i][j]);
   }
   }
   /*  Liberación de la memoria asignada a las filas */
   for(i=0;i<F;i++)
   free(A[i]);
/*Liberación de la memoria asignada al vector de punteros */
   free(A);
   }
```

Si se ejecuta el programa con diferentes datos para la pareja (F, C), se tendría:

F=2 C=3	F=3 C=4	
A[0][0]= 0	A[0][0]= 0	A[1][2]= 3
A[0][1]= 1	A[0][1]= 1	A[1][3]= 4
A[0][2]= 2	A[0][2]= 2	A[2][0]= 2
A[1][0]= 1	A[0][3]= 3	A[2][1]= 3
A[1][1]= 2	A[1][0]= 1	A[2][2]= 4
A[1][2]= 3	A[1][1]= 2	A[2][3]= 5

5.10 Enunciados de las prácticas

Ejercicio 1.

Ejecutar el siguiente programa C interpretando la salida a pantalla obtenida:

```
#include <stdio.h>

void main(  )
 {
          char  *cp,ch;
         int     *ip,i;
          float  *fp,f;
          double  *dp,d;
          cp=&ch ;
         ip=&i ;
         fp=&f;
         dp=&d;

/* Imprime los valores actuales */

         printf("%p %p %p %p \n",cp,ip,fp,dp);

/* Ahora los incrementa en una unidad */

         cp++;
         ip++;
         fp++;
         dp++;

/* Imprime los nuevos valores */

         printf("%p %p %p %p \n",cp,ip,fp,dp);
}
```

Ejercicio 2.

Determinar la salida a pantalla del siguiente programa C:

```
#include <stdio.h>

void main(  )
{
char *cadena="Altoorlaar a itroxdeous";
char *pt;
int x;
pt=cadena+1;

for(x=0 ;x<12 ;x++)
{
     printf("%c",*pt);
     pt+=2;
}
printf("\n");
}
```

Ejercicio 3.

Determinar la salida a pantalla del siguiente programa C:

```
#include <stdio.h>
void main(  )
{
    char *ptr="Lenguaje  C";
    printf("%s",ptr);
/* Tratamiento de la cadena carácter a carácter */
    while(*ptr)
         printf("%c", *ptr++);
}
```

Ejercicio 4.

Utilizando punteros y un vector de tipo entero que tenga por nombre **year**, escribir un programa C que genere el siguiente resultado:

El mes 1 tiene 31 dias.
El mes 2 tiene 28 dias.
.....
El mes 12 tiene 31 dias.

Ejercicio 5.

Hallar la salida a pantalla del programa C indicado a continuación:

```
#include <stdio.h>

void main( )
{
   int i, a[5]={10,9,8,7,6};
   int b[5]={1,2,3,4,5};
   int *p1,*p2 ;
   p1=&a[4] ;
   p2=b ;
   for(i=0;i<5;i++)
       *p2++=*p1--;
   for(i=0;i<5;i++)
        printf("a[%d]=%d b[%d]=%d \n",i,a[i],i,b[i]);
}
```

Ejercicio 6.

En el caso de cadenas, la doble indirección da acceso a cada uno de los elementos que las componen. Como ejemplo de esto, hallar la salida del siguiente programa C:

```
#include <stdio.h>

void  main( )
{
    char *ptr[ ]={ "Lenguaje C", "Lenguaje C++"};
    int v1,v2;
    for(v1=0;v1<2;v1++)
   {
        for(v2=0;v2<13;v2++)
                printf("%c", *(*(ptr+v1)+v2));
        printf("\n");
     }

    for(v1=0;v1<2;v1++)
    {
       v2=0;
       while( (*(*(ptr+v1)+v2)) != '\0')
       {
            printf("%c", *(*(ptr+v1)+v2));
            v2++;
        }
       printf("\n");
    }
}
```

Ejercicio 7.

Utilizando doble indirección, diseñar un programa C que calcule la traspuesta de la matriz:

$$A = \begin{matrix} 1 & -1 & 5 \\ 2 & 0 & 1 \\ -7 & 3 & 0 \end{matrix}$$ y se la sume a la matriz $$B = \begin{matrix} 1 & 2 & 7 \\ 1 & 3 & 4 \\ 5 & -1 & 1 \end{matrix}$$

Las matrices deben ser inicializadas al declararse.

Ejercicio 8.

Escribir un programa que muestre una tabla con las 4 primeras potencias de los números de 1 a n en la siguiente forma :

N	N^2	N^3	N^4
1	1	1	1
2	4	8	16
3	9	27	81
.....			
n	n^2	n^3	n^4

El programa leerá por teclado n y realizará el dimensionamiento dinámico de la matriz P(n,4) que contendrá los resultados de la tabla.

5.11 Soluciones a las prácticas del capítulo 5

Ejercicio 1.

La salida a pantalla es la siguiente:

0065FDF0	0065FDE8	0065FDE0	0065FDD4
0065FDF1	0065FDEC	0065FDE4	0065FDDC

En la primera línea aparecen las direcciones de memoria de las variables ch, i, f y d, es decir, los punteros que apuntan a ellas. Posteriormente se incrementan los punteros, es decir, pasan a apuntar al siguiente í*tem* según su tipo. Las nuevas direcciones de memoria son las anteriores sumadas 1 byte, para el tipo char, 4 bytes, para los tipos int y float y 8 bytes para el tipo double.

Ejercicio 2.

Si se considera fijo el valor inicial de pt, se cumple lo siguiente:

```
*pt=l
*(pt+2)=o
*(pt+4)=r
*(pt+6)=a
*(pt+8)=r
```

y así sucesivamente, por tanto, la salida es:

lorarairxeu

Ejercicio 3.

La salida obtenida al ejecutar el programa es la siguiente:

Lenguaje CLenguaje C

Ejercicio 4.

Una solución al ejercicio es la que se indica a continuación:

```
#include <stdio.h>
void main( )
{
    int year[ ]={31,28,31,30,31,30,31,31,30,31,30,31};
    int *ptr,vcb=0;
    ptr=year ;
    while(vcb<12)
        printf("El mes %d tiene %d dias \n", (vcb++)+1, *ptr++);
}
```

Nota: Aquí `(vcb++)+1` funciona de la siguiente forma: primero se imprime `vcb+1` y luego se hace el incremento `vcb++ (vcb=vcb+1)`. La expresión `*ptr++` evalúa primero `*ptr` (que es lo que se imprime) y posteriormente `ptr=ptr+1`.

Ejercicio 5.

En primer lugar hay que notar, que la expresión `*p2++=*p1- -`, equivale a:

```
*p2=*p1;
p1=p1-1;
 p2=p2+1;
```

La salida a pantalla es la siguiente:

a[0]=10	b[0]=6
a[1]=9	b[1]=7
a[2]=8	b[2]=8
a[3]=7	b[3]=9
a[4]=6	b[4]=10

Ejercicio 6.

A continuación se indica la salida a pantalla:

Lenguaje C
Lenguaje C++
Lenguaje C
Lenguaje C++

Ejercicio 7.

Un programa válido es el siguiente:

```
#include <stdio.h>

void main( )
{
    int i,j;
    int A[3][3]={1,-1,5,2,0,1,-7,3,0};
    int B[3][3]={1,2,7,1,3,4,5,-1,1};
    int C[3][3];
    for(i=0;i<3;i++)
      for(j=0;j<3;j++)
      {
         *(*(C+i)+j)=*(*(A+j)+i);
         *(*(C+i)+j)=*(*(C+i)+j)+ *(*(B+i)+j);
         printf("C[%d][%d]=%d \n", i,j, *(*(C+i)+j));
      }
}
```

Ejercicio 8.

A continuación se ofrece una solución válida:

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

void main(  )
{
    int i,j,n, **p;
    printf("Introduce el numero de filas de la tabla \n ");
    scanf("%d", &n);
/* Asignación de memoria para un vector de punteros */
    p=malloc(n*sizeof(int*));
    if(p==NULL)
   {
        printf("Error de localización de memoria ");
        exit(0);
    }
/* Asignación de memoria para cada una de las filas de la matriz */
    for(i=0;i<n;i++)
    {
        p[i]=malloc(4*sizeof(int));
        if (p[i]== NULL)
       {
           printf("Error de localización de memoria ");
           exit(0);
       }
    }
/* más abajo, se ofrece una solución alternativa al próximo bucle */
    for(i=0;i<n;i++)
    {
        for(j=0;j<4;j++)
        {
           if(j == 0)         p[i][j]=i+1;
           else  if(j == 1)   p[i][j]=(i+1)*(i+1);
           else  if(j == 2)   p[i][j]=(i+1)*(i+1)*(i+1);
           else               p[i][j]=(i+1)*(i+1)*(i+1)*(i+1);
        }
     }
```

```
    printf("%10s%10s%10s%10s \n ", "N", "N^2", "N^3", "N^4");
    for(i=0;i<n;i++)
    {
        for(j=0;j<4;j++)
                printf("%10d", p[i][j]);
        printf("\n");
     }
}
```

Una solución alternativa al bucle indicado es la siguiente:

```
/* se debe declarar x como variable entera*/
for(i=0; i<n; i++)
{
        x=1;
        for(j=0;j<4;j++)
        {
              x=x*(i+1);
              p[i][j]=x;
        }
}
```