Punteros

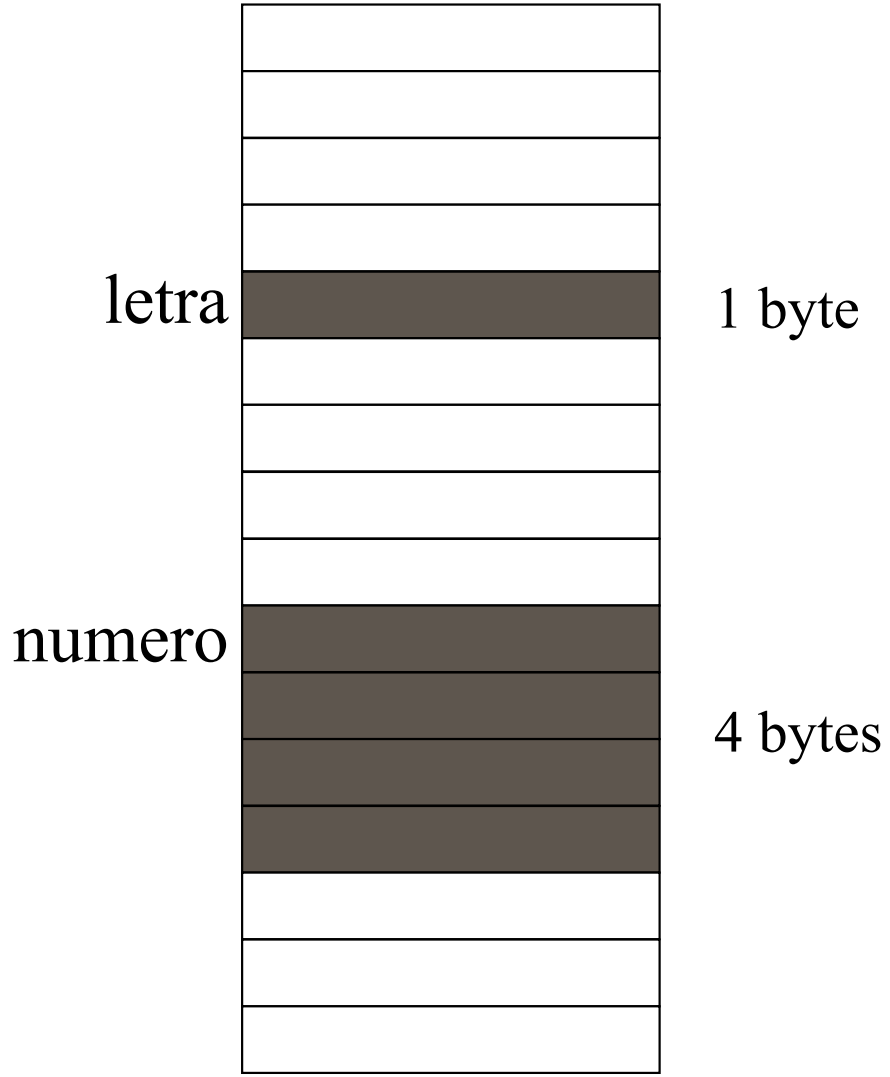
- Recordatorio: La memoria RAM de un ordenador está compuesta por un gran número de celdas de información (bytes)
- Cada celda tiene un número de identificación que la distingue del resto
  - *Este número es la dirección de memoria y sirve para acceder a una celda específica*

Punteros

- Una variable es una zona de memoria reservada para almacenar un valor concreto perteneciente a un tipo de dato
  - *Cualquier variable ocupa un numero entero de bytes*
  - *Cuando se accede a una variable almacenada en memoria, el compilador necesita disponer de los siguientes datos*
    - Número de bytes que componen la variable
      - Viene definido por el tipo de datos de dicha variable
      - El compilador se encarga de reservar, para cada variable, el número de bytes que indique el tipo.
    - Dirección de memoria del byte inicial de la variable
      - La dirección de memoria correspondiente al byte inicial de una variable viene representada por el nombre de dicha variable
      - El compilador es el encargado de sustituir el nombre de la variable por su dirección de memoria

Punteros

- char letra
- long numero



Punteros

- Operador de dirección : '&'
  - *Para conocer la dirección de una variable, C incorpora el operador de dirección (&)*

```
void main()
{
float n,array[10];
clrscr();
printf("direccion de variable n = %p\t%lu\n",&n,&n);
getch();
}
```

Especificador para imprimir punteros

Un computador de 32 bits almacena direcciones en 4 bytes

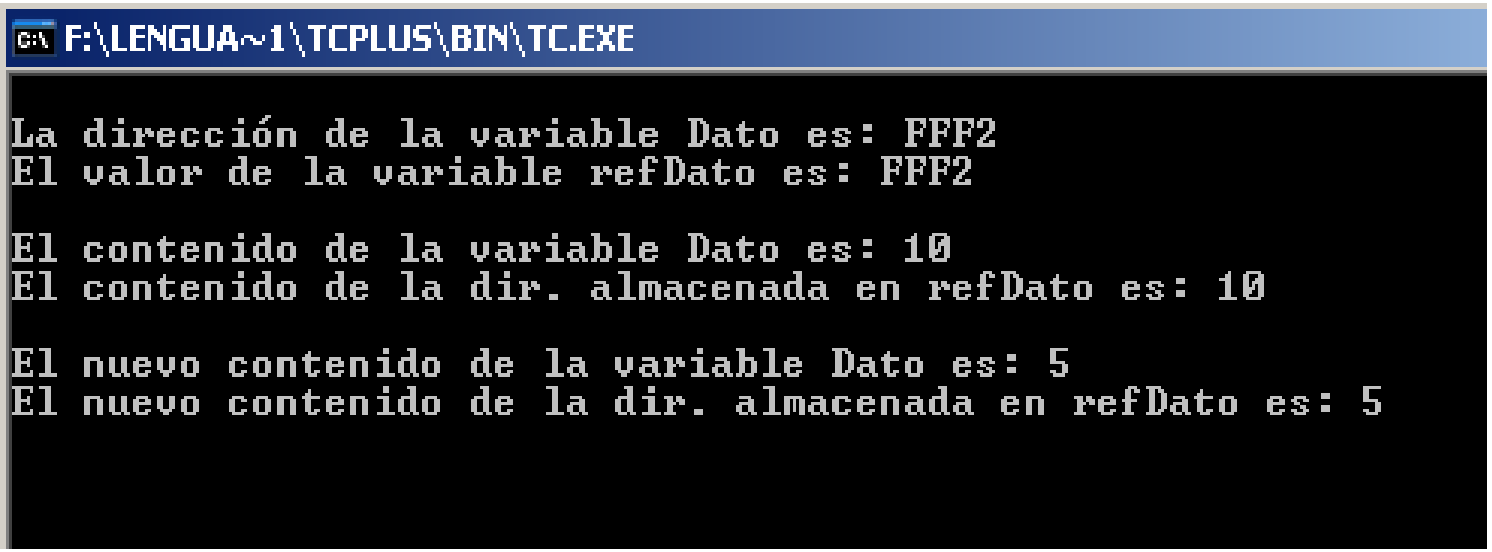
- El operador "&" sólo se puede aplicar a variables y a elementos de un array
  - &(x+1) o &3 es ilegal

Punteros

- Operador de indirección
- El operador * toma su operando como una dirección y accede a esa dirección para obtener su contenido
  - *y=*px asigna a "y" el valor contenido en la dirección donde apunte px*
  - *px=&x*
  - *y=*px* → y=x
  - *El operador de indirección efectúa la operación opuesta al operador de dirección*
    - Este operador no se puede aplicar a ninguna variable que no sea de tipo puntero

Punteros

```
#include <stdio.h>
int main()
{
long Dato = 0;
long *refDato;
clrscr();
refDato = &Dato;
printf("\nLa direcci¢n de la variable Dato es: %p\n", &Dato);
printf("El valor de la variable refDato es: %p\n", refDato);
Dato = 10;
printf("\nEl contenido de la variable Dato es: %ld\n", Dato);
printf("El contenido de la dir. almacenada en refDato es: %ld\n", *refDato);
*refDato = 5;
printf("\nEl nuevo contenido de la variable Dato es: %ld\n", Dato);
printf("El nuevo contenido de la dir. almacenada en refDato es: %ld\n", *refDato);
getch();
return(0);
}
```



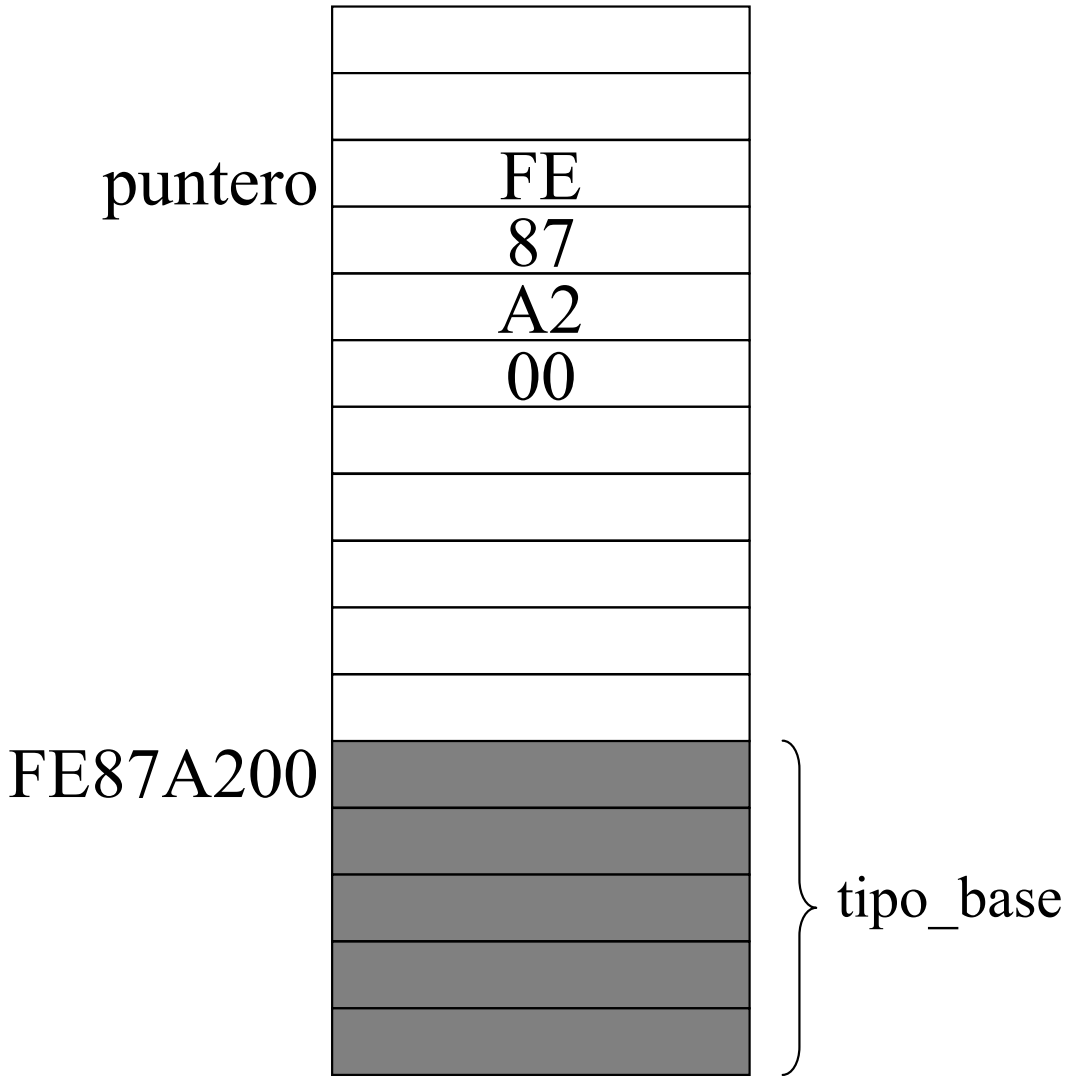
Punteros

- Atención:
  - *Una vez se ha asignado la dirección de una variable a una variable puntero, existen dos formas de acceder al contenido de dicha variable:*
    - A través del nombre de la variable
    - A través del operador de indirección sobre el puntero
  - *¿Qué ocurre si se utiliza el operador de indirección sobre una variable puntero no inicializada?*
    - Una variable puntero sin inicializar contendrá un valor aleatorio de la memoria. Si aplicamos el puntero de indirección accedemos a una zona de memoria desconocida que:
      - Podría ser de acceso prohibido
      - Podría ser la dirección de una variable importante del programa
      - Este tipo de error no se detecta en la compilación

Punteros

- Un puntero es una variable que contiene la dirección de otra variable
  - *Los punteros se utilizan mucho en C*
    - A veces son la única manera de expresar un cálculo
    - El código es más compacto
- La sintaxis de un puntero es:
- tipo_base *variable
  - *El tipo_base puede ser cualquier tipo predefinido en C o cualquiera definido por el usuario (typedef)*
  - *La variable definida podrá almacenar una dirección de memoria y los datos que se almacenan en dicha dirección serán tratados como datos del tipo_base*
  - *El tamaño en bytes de una variable puntero viene dado por el tamaño de dirección que usa el computador*
    - Un computador de 32 bits suele utilizar punteros de 4 bytes

Punteros



Punteros

```
#include <math.h>
void main()
{
int x, *px, *py;
clrscr();
x = 5;
px = &x;
printf("%d\t%f",*px+1, sqrt((float)*px+4));
*px=0;
printf("\n%d",x);
(*px)++;
py=px;
getch();
}
```

Los punteros pueden aparecer en expresiones

cast

Referencias a apuntadores en la parte izquierda de la asignación

Se necesitan los paréntesis porque los operadores unitarios se evalúan de derecha a izquierda

py apunta a donde apunta px

Punteros

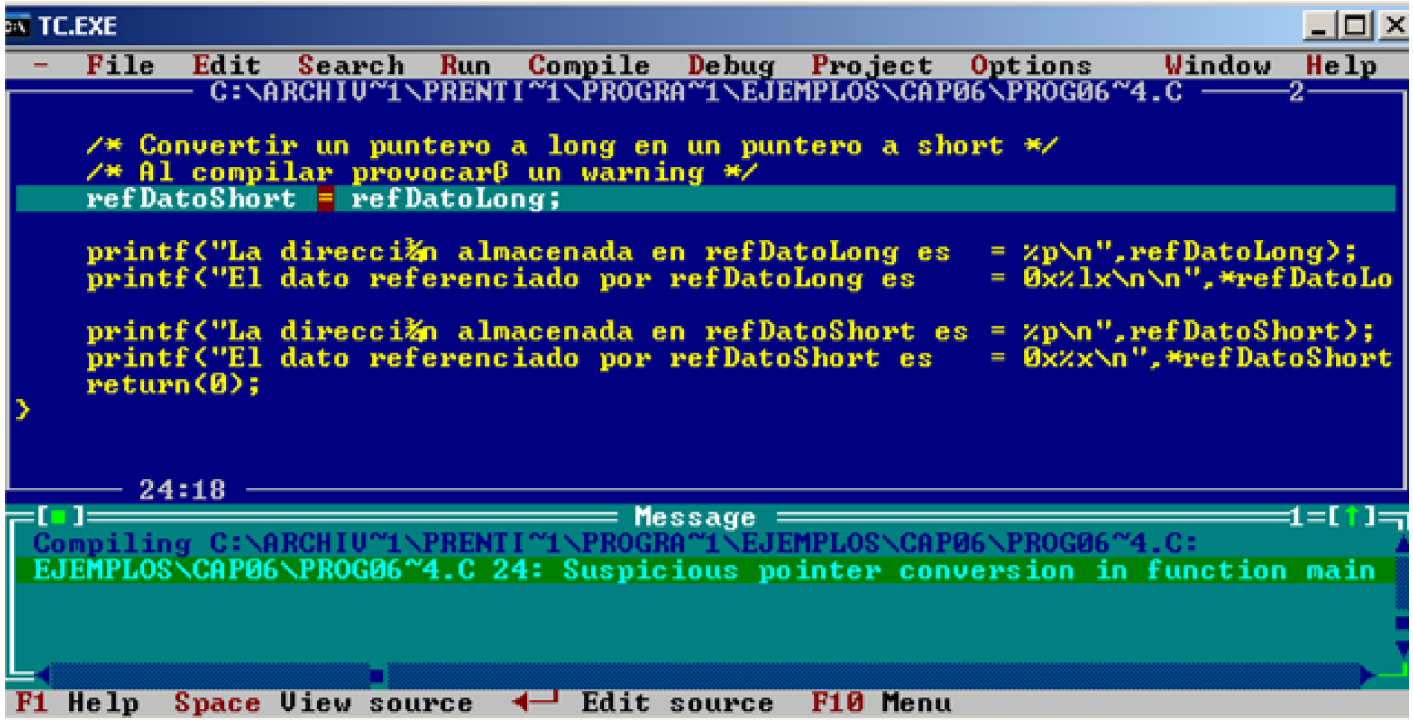
- Operaciones con punteros
  - *Conversión de punteros*
    - Una variable de tipo puntero tiene un tipo_base que sirve de patrón de tamaño cuando se accede al dato referenciado por el puntero.
    - Sin embargo no es obligatorio que ese dato pertenezca o deba pertenecer al tipo_base
    - Esto permite utilizar los datos de forma muy flexible y poco ortodoxa, aunque por otro lado puede ser una fuente de errores no controlados.

Punteros

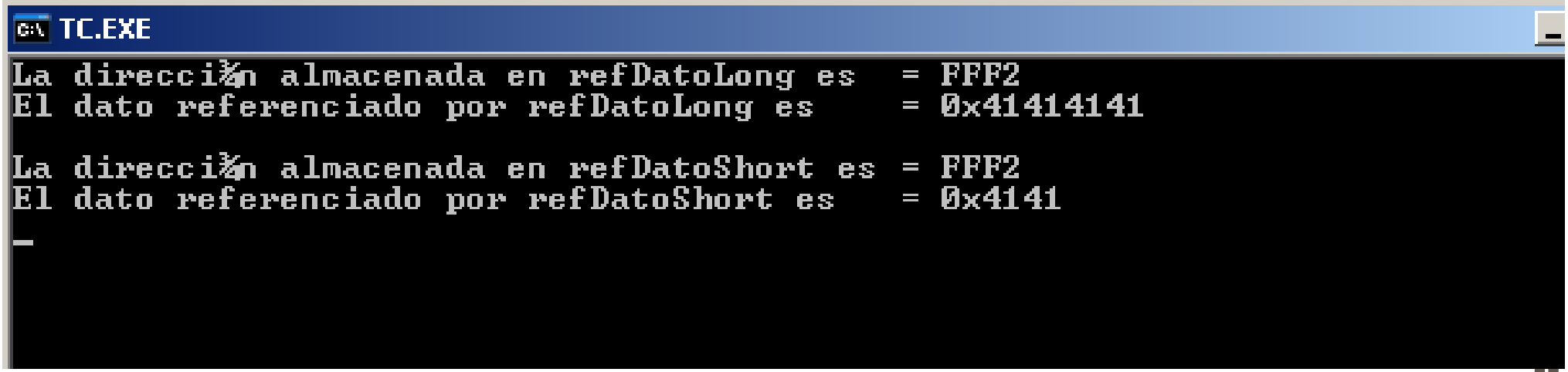
```
#include <stdio.h>

int main(void)
{
    long Dato = 0x41414141;
    long * refDatoLong;
    short * refDatoShort;

    /* Asignar la dirección de un long al puntero a long */
    refDatoLong = &Dato;

    /* Convertir un puntero a long en un puntero a short */
    /* Al compilar provocará un warning */
    refDatoShort = refDatoLong;

    printf("La dirección almacenada en refDatoLong es  = %p\n",refDatoLong);
    printf("El dato referenciado por refDatoLong es    = 0x%lx\n\n",*refDatoLong);

    printf("La dirección almacenada en refDatoShort es = %p\n",refDatoShort);
    printf("El dato referenciado por refDatoShort es   = 0x%x\n",*refDatoShort);
    return(0);
}
```

En la mayoría de lenguajes esta asignación está prohibida. C lo consiente, avisándote en el proceso de compilado, que hay una asignación sospechosa.

Punteros



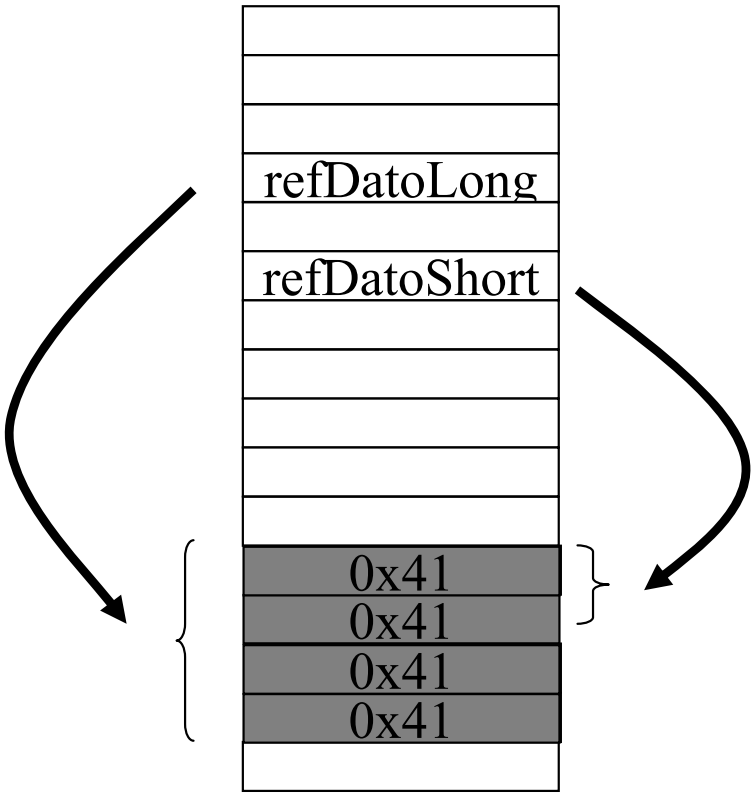
El tipo long ocupa 4 bytes
El tipo short ocupa 2 bytes



Punteros

- Si queremos evitar que el compilador nos de estos avisos, hay que realizar una conversión de tipos "cast":

```
refDatoShort = (short *) refDatoLong;
```



Aritmética de punteros

- La aritmética de punteros consiste en tratar a los punteros como lo que son, direcciones de memoria.
- Cuando se trabaja en ensamblador, es posible
  - *asignar direcciones de memoria a variables*
  - *utilizar estas variables para acceder a los datos almacenados*
  - *Incrementar o decrementar esas variables para acceder a direcciones contiguas a la original.*
- Con la aritmética de punteros hemos de ser capaces de realizar lo mismo

Aritmética de punteros

- Con la aritmética de punteros añadimos o sustraemos valores enteros a una variable de tipo puntero para hacer referencia a direcciones anteriores o posteriores a la almacenada originalmente en el puntero.
  - *Cuando se hace esta operación, lo que realmente se hace es incrementar o decrementar la dirección en una cantidad referenciada al tamaño en bytes del tipo base*
    - Imaginemos un puntero char: `Char *refDatoChar`

      `refDatoChar = refDatoChar + 1`

      es una expresión válida que indica que, si refDatoChar contenía la dirección 1000, después de la operación contiene la 1001 :

      dirección original + 1*sizeof(char) = 1000 + 1*1 = 1001
    - Imaginemos un puntero long: long `*refDatoLong`

      `refDatoLong = refDatoLong + 1`

      dirección original + 1*sizeof(long) = 1000 + 1*4 = 1004

Punteros

- Puntero nulo
  - *Para minimizar los daños que pueda ocasionar la practica de no inicializar los punteros, hay que asegurarse que los punteros que no hayan recibido un valor útil, contengan un valor lo mas inocuo posible.*
    - Así, si se accede a ellos de forma involuntaria, el error será predecible.
  - *Para conseguir este efecto es necesario seleccionar una zona de memoria neutra: la dirección 0.*
    - Por convenio, es imposible que la dirección de memoria 0 contenga datos válidos relacionados con el programa. El compilador está preparado para que cualquier acceso a dirección 0 de un error de ejecución, terminando el programa en curso
- Para facilitar esta dirección, se utiliza una constante expresamente reservada y definida en varias librerías (por ejemplo en la stdio.h): "NULL"

```
#include <stdio.h>

int main(void)
{

    long * refDatoLong = NULL;
    long aux;

    printf("La dirección almacenada en refDatoLong = %p\n", refDatoLong);

    /* Comprobar si el puntero es nulo */
    if (NULL == refDatoLong) {
        printf("El puntero refDatoLong es nulo\n");
    }
    /*Prueba de acceso al contenido de refDatoLong, debería dar error */
    printf ("Acceso al contenido de refDatoLong, debe de producirse un error");
    aux = *refDatoLong;

    return(0);
}
```

Permite el uso de NULL

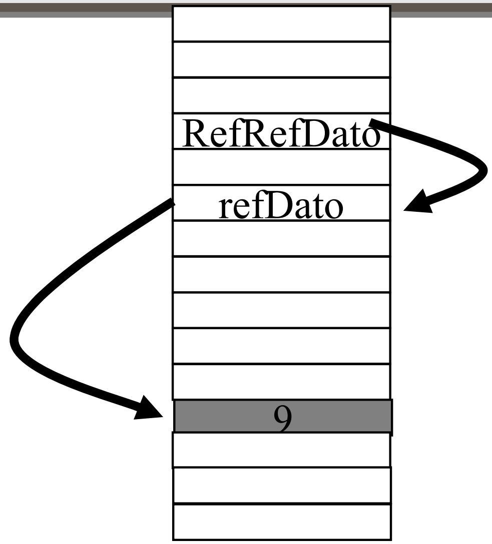
Long *refDatoLong es una definición válida pero deja a la variable puntero con valor indefinido. Hasta el momento de la asignación, esta variable es un peligro

Punteros a punteros

- Puesto que el tipo base puede ser cualquier tipo, este también puede ser el tipo puntero
- Por lo tanto se pueden definir punteros a punteros:

```
long * * ptr
```

  - *Define una variable llamada ptr de tipo puntero a puntero a long*
  - *En este caso, ptr almacena direcciones de memoria en las que a su vez se almacenan direcciones de memoria de tipo long*




```c
#include <stdio.h>

int main(void)
{

    long Dato = 9;
    long *refDato;
    long **refRefDato;

    refDato = &Dato;
    refRefDato = &refDato;

    printf("La dirección almacenada en refDato es %p\n", refDato);
    printf("La dirección almacenada en refRefDato es %p\n", refRefDato);

    printf("El valor almacenado en Dato es %ld\n", *refDato);
    printf("El valor almacenado en refDato es %p\n", *refRefDato);

    printf("El valor almacenado en Dato es %ld\n", **refRefDato);

    return(0);
}
```

Punteros

- Ya que C pasa por valor los argumentos de las funciones, no hay forma directa de que la función llamada altere un variable de la función que llama

```
/* Paso por valor. */
#include <stdio.h>
void intercambio(int,int);
main() /* Intercambio de valores */
{
int a=1,b=2;
clrscr();
printf(" a=%d y b=%d",a,b);
intercambio(a,b); /* llamada */
printf(" a=%d y b=%d",a,b);
}

void intercambio (x,y)
int x;
int y;
{
int aux;
aux=x;
x=y;
y=aux;
printf(" a=%d y b=%d",x,y);
}
```

Punteros

- Pasando punteros como argumentos podemos conseguir el efecto deseado: intercambiar realmente el contenido de las variables

```
/* Paso por referencia. */
#include <stdio.h>
void intercambio(int *,int *);
main() /* Intercambio de valores */
{
int a=1,b=2;
clrscr();
printf(" a=%d y b=%d",a,b);
intercambio(&a,&b); /* llamada */
printf(" a=%d y b=%d",a,b);
}

void intercambio (int *x,int *y)
{
int aux;
aux=*x;
*x=*y;
*y=aux;
printf(" a=%d y b=%d",*x,*y);
}
```

Punteros

- Retorno de mas valores
  - *El paso de valores por referencia permite que una función pueda devolver más de un valor.*
  - *Generar un programa que devuelva el cálculo de la suma resta multiplicación y división de dos números.*