

Gestión de memoria dinámica

martes, 8 de octubre de 2024 4:25 p. m.



Heap

Memoria estática

- Reserva antes de la ejecución del programa
- Permanece fija
- No requiere gestión durante la ejecución
- El sistema operativo se encarga de la reserva, recuperación y reutilización.
- Variables globales y static.

The diagram shows a vertical stack of memory segments. At the top is a dashed-line box labeled 'Segmento de código'. Below it is a solid-line box labeled 'Memoria Estática'. The next section is labeled 'Heap Montón'. A downward-pointing arrow is positioned to the right of this section. The bottom section is labeled 'Espacio Libre'. An upward-pointing arrow is positioned to the left of this section. The final section at the bottom is labeled 'Pila Stack'.

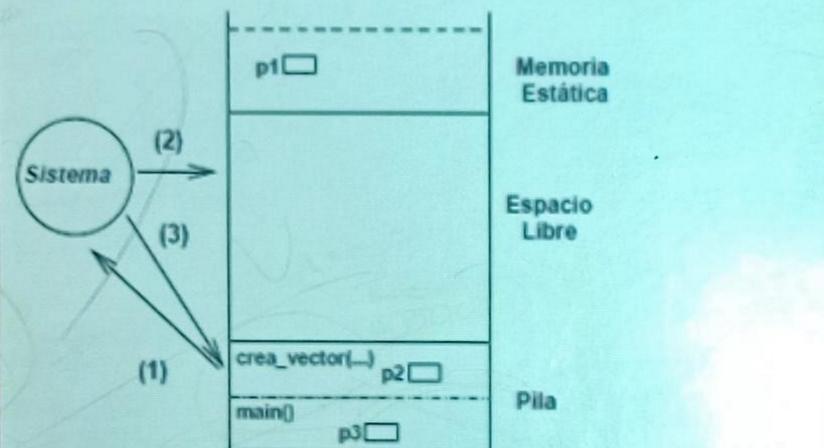
HEAP

Es una zona de memoria donde se reservan y se liberan "trozos" durante la ejecución de los programas según sus propias necesidades. Esta memoria surge de la necesidad de los programas de "crear nuevas variables" en tiempo de ejecución con el fin de optimizar el almacenamiento de datos.

EJEMPLO

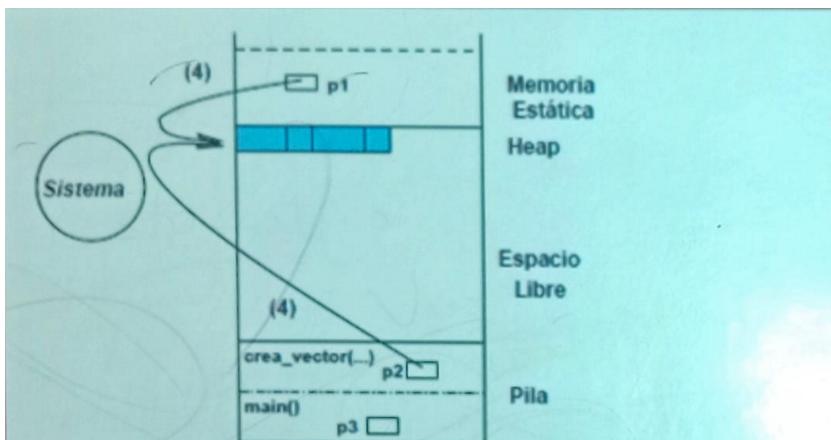
(ejemplo)

El sistema operativo es el encargado de controlar la memoria que queda libre en el sistema.



- (1) Petición al S.O. (tamaño)
- (2) El S.O. comprueba si hay suficiente espacio libre.
- (3) Si hay espacio suficiente, devuelve la ubicación donde se encuentra la memoria reservada, y marca dicha zona como memoria ocupada.

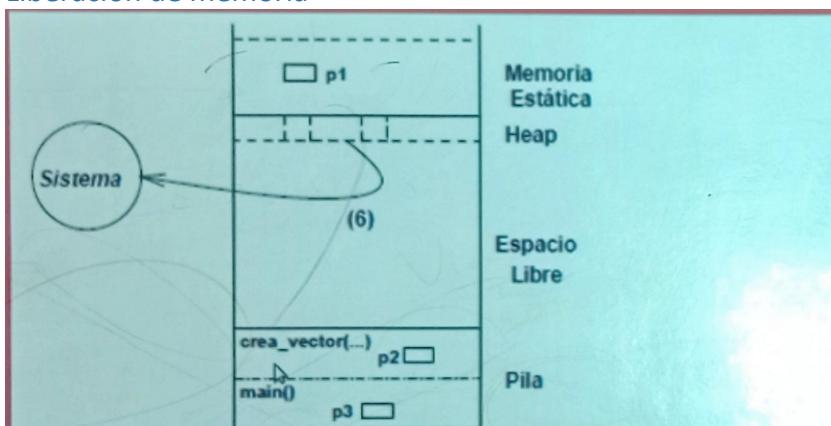
Reserva de memoria



(4) La ubicación de la zona de memoria se almacena en una variable estática (**p1**) o en una variable automática (**p2**).

Por tanto, si la petición devuelve una dirección de memoria, **p1** y **p2** deben ser variables de tipo *puntero* al tipo de dato que se ha reservado.

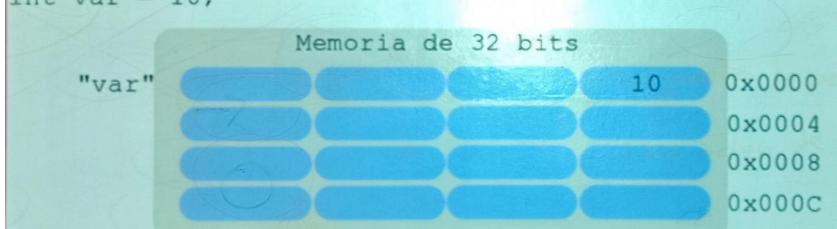
Liberación de memoria



(6) Finalmente, una vez que se han utilizado las variables dinámicas y ya no se van a necesitar más es necesario liberar la memoria que se está utilizando e informar al S.O. que esta zona de memoria vuelve a estar libre para su utilización.

Puntero

```
int var = 10;
```



Al usar punteros se está referenciando ubicaciones de memoria

Permiten manipular la memoria de forma efectiva

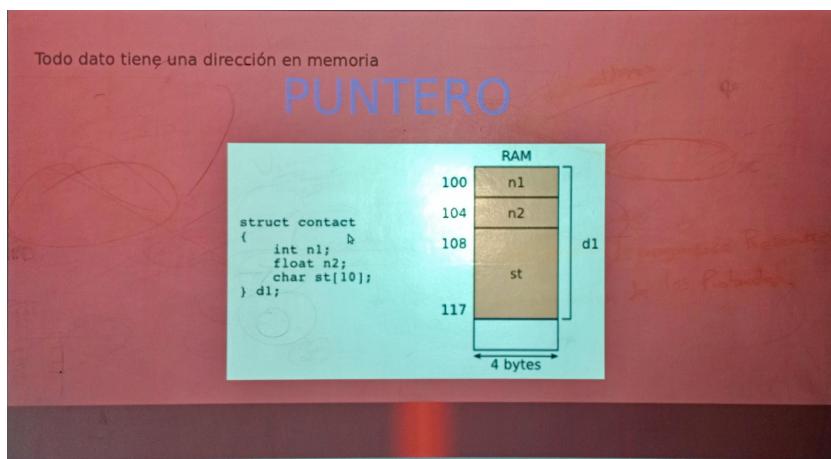
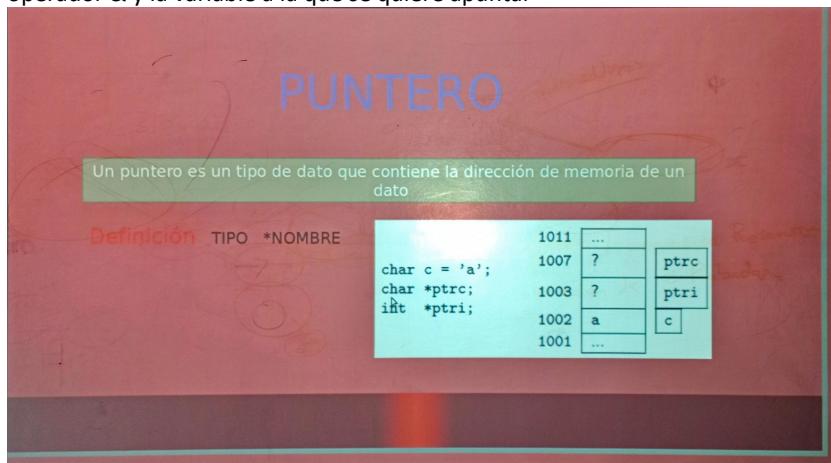
- El tamaño de todas variables y su posición en memoria.
- Todo dato está almacenado a partir de una dirección de

Tipo T	Tamaño (bytes) [a]	Puntero a T	Tamaño (bytes)	Ejemplo de uso
int	4	int *	4	int *a, *b, *c;
unsigned int	4	unsigned int *	4	unsigned int *d, *e, *f;
short int	2	short int *	4	short int *g, *h, *i;
unsigned short int	2	unsigned short int *	4	unsigned short int *j, *k, *l;
long int	4	long int *	4	long int *m, *n, *o;
unsigned long int	4	unsigned long int *	4	unsigned long int *p, *q, *r;
char	1	char *	4	char *s, *t;
unsigned char	1	unsigned char *	4	unsigned char *u, *v;
float	4	float *	4	float *w, *x;
double	8	double *	4	double *y, *z;
long double	8	long double *	4	long double *a1, *a2;

Un puntero es una variable que contiene la dirección de otra variable.

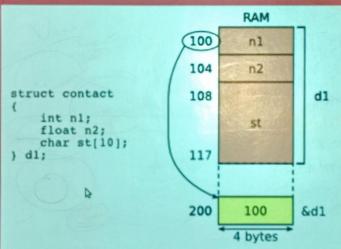
```
int *nombre; // apint
float *nombre; // apfloat
ClaseA *nombre; apClaseA
```

Para apuntar un puntero a una variable se utilizan el operador de asignación = , el operador & y la variable a la que se quiere apuntar



La Indirección

PUNTERO



Memoria		Memoria		Memoria		Memoria	
100	????	100	????	100	10	100	10
104	????	104	????	104	20	104	20
108	????	108	100	108	100	108	104
112	????	112	104	112	104	112	NULL

Antes de ejecutar la línea 6

Tras ejecutar la línea 7

Tras ejecutar la línea 10

Tras ejecutar la línea 13

```
1 #include <stdio.h>
2 int main(int argc, char** argv)
3 {
4     int num1, num2;
5     int *ptr1, *ptr2;
6
7     ptr1 = &num1;
8     ptr2 = &num2;
9
10    num1 = 10;
11    num2 = 20;
12
13    *ptr1 = 30;
14    *ptr2 = 40;
15
16    *ptr2 = *ptr1;
17
18    return 0;
19 }
```

https://www.it.uc3m.es/pbsanta/asng/course_notes/pointers_data_type_es.html

Terminal mar 17:35

```
#include<stdio.h>
double numero ;
int main(){
//static double otronumero = 321;
return 0;
}

Archivo Editar Ver Buscar Terminal Ayuda
-rw-rwxrwx 1 jvcc jvcc 30527 nov 30 2021 v1.pdf
drwxrwxrwx 5 jvcc jvcc 4896 sep 9 18:20 Memoria
jvcc@jvcc-ThinkPad-T490s:~/Documentos/SSOO/SISTEMAS OPERATIVOS/Memoria$ cd Memoria/
jvcc@jvcc-ThinkPad-T490s:~/Documentos/SSOO/SISTEMAS OPERATIVOS/Memoria$ dir
ejerciclo1.cpp Memoria1.exe Puntero15.cpp Puntero3.exe Puntero9.cpp
ejerciclo1.exe Puntero10.cpp Puntero15.exe Puntero4.cpp vector.cpp
napa1.c Puntero11.cpp Puntero11.exe Puntero4.exe vector.exe
napa1.c Puntero12.cpp Puntero12.exe Puntero5.cpp
napa2.c Puntero13.cpp Puntero12.cpp Puntero6.cpp
napa2.c Puntero14.cpp Puntero2.exe Puntero7.cpp
Memoria1.cpp Puntero14.exe Puntero3.cpp Puntero8.cpp
jvcc@jvcc-ThinkPad-T490s:~/Documentos/SSOO/SISTEMAS OPERATIVOS/Memoria$ ./napa2
jvcc@jvcc-ThinkPad-T490s:~/Documentos/SSOO/SISTEMAS OPERATIVOS/Memoria$ ./napa2

jvcc@jvcc-ThinkPad-T490s:~/Documentos/SSOO/SISTEMAS OPERATIVOS/Memoria$ gcc napa1.c -o mapal && size mapal
text data bss dec hex filename
1415 544 8 1967 7af mapal
jvcc@jvcc-ThinkPad-T490s:~/Documentos/SSOO/SISTEMAS OPERATIVOS/Memoria$ gcc mapal1.c -o mapal1 && size mapal1
text data bss dec hex filename
1415 544 16 1975 7b7 mapal1
jvcc@jvcc-ThinkPad-T490s:~/Documentos/SSOO/SISTEMAS OPERATIVOS/Memoria$
```

C mapa1.c ●

C mapa1.c > ...

```
1 #include <stdio.h>
2 //double numero = 123;
3 int main(){
4 //static double otronumero 321 ;
5 return 0;
6 }
```

```
nik@LAPTOP-TPHC0EUP:~/so/ejercicios$ gcc mapal.c -o mapal && size mapal
text data bss dec hex filename
1228 552 8 1788 6fc mapal
```

C mapa1.c X

C mapa1.c > [e] numero

```
1 #include <stdio.h>
2 double numero;
3 int main(){
4 //static double otronumero 321 ;
5 return 0;
6 }
```

```
nik@LAPTOP-TPHC0EUP:~/so/ejercicios$ gcc mapal.c -o mapal && size mapal
text data bss dec hex filename
1228 544 16 1788 6fc mapal
```

```
C mapa1.c X
C mapa1.c > ...
1 #include <stdio.h>
2 double numero = 123;
3 int main(){
4 //static double otronunero 321 ;
5 return 0;
6 }
7
```

```
nik@LAPTOP-TPHC0EUP:~/so/ejercicios$ gcc mapal.c -o mapal && size mapal
text      data      bss      dec      hex filename
1228      552       8     1788      6fc mapal
```

```
C mapa1.c > ⌂ main()
1 #include <stdio.h>
2 double numero = 123;
3 int main(){
4 static double otronunero;
5 return 0;
6 }
```

```
nik@LAPTOP-TPHC0EUP:~/so/ejercicios$ gcc mapal.c -o mapal && size mapal
text      data      bss      dec      hex filename
1228      552      16     1796      704 mapal
```

```
C mapa1.c X
C mapa1.c > ⌂ main()
1 #include <stdio.h>
2 double numero = 123;
3 int main(){
4 static double otronunero = 321;
5 return 0;
6 }
```

```
nik@LAPTOP-TPHC0EUP:~/so/ejercicios$ gcc mapal.c -o mapal && size mapal
text      data      bss      dec      hex filename
1228      560       8     1796      704 mapal
```

```
C mapa2.c X
C mapa2.c > [e] cadena
1 #include <stdio.h>
2
3 char cadena[8192*10] = 'hola';
4
5 ↘ int main(){
6   |   getchar();
7   |   return 0;
8 }
```

Pmap -x (proceso)

Pmap -x 8459