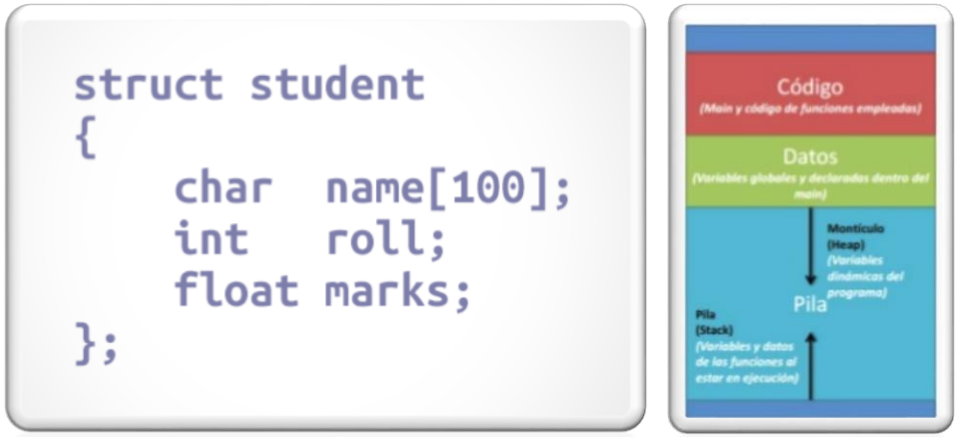
[Organización de Computadoras]



MANEJO DE REGISTROS EN C.

ADMINISTRACIÓN DE MEMORIA EN C.

Copyright

Complemente este documento con otros recursos online

martes, 13 de septiembre de 2022

Enlaces externos de interés

- Acceda a información útil mediante enlaces externos a:
  - **código fuente** disponible de forma **online**.
  - **otro material** disponible de forma **online**.
- Un **video tutorial** sobre los temas que aborda este documento puede encontrarse en:

Manejo de Registros en C

Repaso registros :: definición

- Los **registros** son un **tipo de dato**, que se compone de un conjunto de **campos** que son de otros tipos, ya sean básicos o complejos.
- Cada **registro** tiene asociado una **etiqueta** que lo identifica.
- A su vez, cada **campo** de un **registro** es definido indicando su **tipo de dato** y una etiqueta que lo identifica.
- Por ejemplo:
  - **alumno** es un registro con 3 campos.
  - **dni** es un campo de tipo **long int**.
  - En el ejemplo, el **struct alumno** fue formalmente **definido**, aunque no hubo en esta instancia **declaración de variables**, ni **reservación** de **memoria**.

```
struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
}
```

Repaso registros :: definición de variables

- Existes **múltiples** maneras de **declarar variables** asociadas a un **registro**.
- Respecto del **momento** en el que son definidas:

Junto con la **definición**

```
struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
} alumno_1, alumnos [200];
```

Luego de la **definición**

```
struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
};

struct alumno alumno_1;
struct alumno alumnos [200];
```

- En ambos casos, **alumno_1** es un **struct alumno**, mientras **alumnos** es un arreglo de 200 componentes del tipo **struct alumno**.

Repaso registros :: definición de variables

- Existes **múltiples** maneras de **declarar variables** asociadas a un **registro**.
- Respecto del **tipo de dato** con el que son definidas:

De tipo **struct**

```
struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
} alumno_2;

struct alumno alumno_3;
```

De tipo **puntero struct**

```
struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
} * alumno_4;

struct alumno * alumno_5;
```

- A los **campos** de las variables **alumno_2** y **alumno_3** se los accede mediante el operador punto (.), mientras a **alumno_4** y **alumno_5** mediante el operador flecha (->).
- Por ejemplo: **alumno_2.dni = 30**, y **alumno_4->dni = 30**.

Repaso registros :: definición de tipos de datos

- La sentencia **typedef** se utiliza para definir **nuevos tipos de datos**, a partir de otros ya definidos.
- La definición de un **tipo de dato** asociado a un **registro**, puede darse:

Junto con la **definición**

```
typedef struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
} tAlumno;
```

Luego de la **definición**

```
struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
};
typedef struct alumno tAlumno;
```

- En ambos casos, **tAlumno** es la forma de declarar un **struct alumno**.
- Por ejemplo, **tAlumno** alumno_1 es equivalente a **struct** alumno alumno_1.

Repaso registros :: definición de tipos de datos

- La sentencia **typedef** se utiliza para definir **nuevos tipos de datos**, a partir de otros ya definidos.
- La definición de un **tipo de dato** asociado a un **registro**, puede darse:

Junto con la **definición**

```
typedef struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
}* tAlumno;
```

Luego de la **definición**

```
struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
};
typedef struct alumno * tAlumno;
```

- En ambos casos, **tAlumno** es la forma de declarar un **puntero a struct alumno**.
- Por ejemplo, **tAlumno** alumno_1 es equivalente a **struct** alumno * alumno_1.

Tipos de datos, registros y reservación de memoria

- Un **error frecuente** cuando se utilizan variables del tipo **puntero**, está asociado con la **correcta forma** en la que debe **reservarse** memoria.
- Consideremos el siguiente ejemplo:

```
typedef struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
} * tAlumno;

struct alumno alumno_1;
tAlumno alumno_2;
```

- ¿Quién determina el **espacio en memoria requerido** para una determinada variable?
  - ✓ El tipo de dato **estático** de la misma.
- ¿Cuánto **espacio en memoria** requiere **alumno_1**?
  - ✓ El tipo **estático** de **alumno_1** es **struct alumno**.
  - ✓ El tamaño del **struct** corresponde a 100*__sizeof__(**char**) + 1*__sizeof__(**long int**).
- ¿Se puede **acceder/modificar** los **campos** del **struct** referenciado por **alumno_1** estando **declarado** e **inicializado** de esta manera?
  - ✓ **Sí**. Al intentar **acceder/modificar** un **campo** de **alumno_1**, se estaría **indexando** la locación de memoria que fue correctamente reservaba para tal fin.

Tipos de datos, registros y reservación de memoria

- Un **error frecuente** cuando se utilizan variables del tipo **puntero**, está asociado con la **correcta forma** en la que debe **reservarse** memoria.
- Consideremos el siguiente ejemplo:

```
typedef struct alumno{
  char nombre [50];
  char apellido [50];
  long int dni;
} * tAlumno;

struct alumno alumno_1;
tAlumno alumno_2;
```

- ¿Quién determina el **espacio en memoria requerido** para una determinada variable?
  - ✓ El tipo de dato **estático** de la misma.
- ¿Cuánto **espacio en memoria** requiere **alumno_2**?
  - ✓ El tipo **estático** de **alumno_2** es **puntero**.
  - ✓ Un **puntero** es representado con un **int** → **sizeof(int)**.
- ¿Se puede **acceder/modificar** los **campos** del **struct** referenciado por **alumno_2** estando **declarado** e **inicializado** de esta manera?
  - ✓ **No**. Al intentar **acceder/modificar** un **campo** de **alumno_2**, se estaría **indexando** una locación de memoria que no fue reservaba para tal fin.

Memoria estática
&
Memoria dinámica

Introducción

- Algunos lenguajes de programación, **son responsables** de llevar a cabo la **gestión de memoria** de forma **automática** y **transparente** al programador.
- La **gestión de memoria** contempla la **asignación** y **liberación** de los espacios de memoria requeridos para el funcionamiento de un programa.
- En **C**, por el contrario, parte de la **gestión de memoria** es responsabilidad del **programador**.

Introducción

- Por contraste, así como en **Java** el programador puede crear **nuevos objetos** sin la necesidad de gestionar memoria, en **C deberá** encargarse tanto de la **reservación** como de la **liberación** de memoria cuando inserte una **nueva celda** en una dada lista.
- En ambos casos, se hace mención a la gestión de **memoria dinámica**, esto es, memoria que se reserva en **tiempo de ejecución**.

Memoria Estática vs. Memoria Dinámica

- Por **memoria estática** referimos al espacio en memoria necesario al declarar variables de cualquier tipo de dato, cuyo tamaño (número de bytes) es conocido en **tiempo de compilación**:
  - **Datos primitivos**: int, char, float.
  - **Datos no primitivos**: struct, array, punteros.
- La memoria que estas variables ocupan no puede **cambiarse** durante la ejecución y tampoco puede ser **liberada manualmente**.

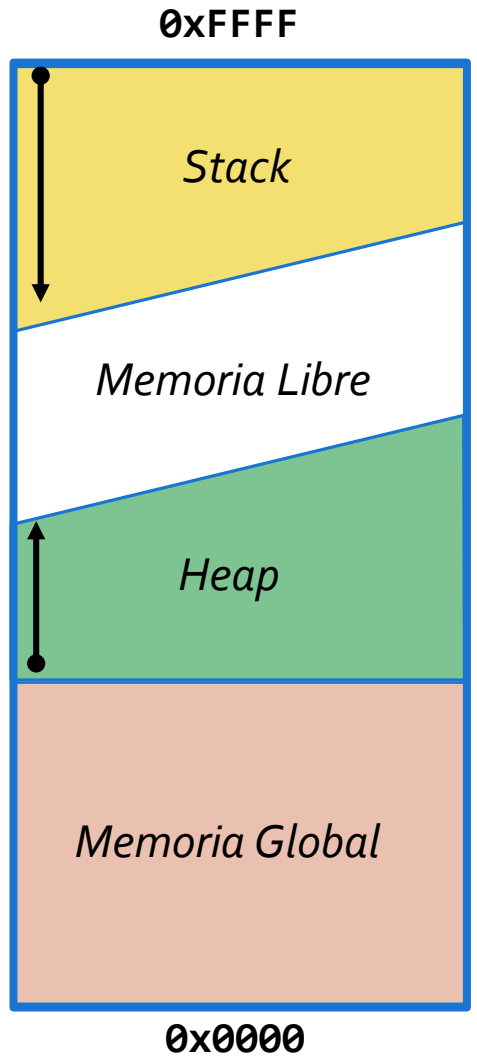
Memoria Estática vs. Memoria Dinámica

- Por **memoria dinámica** referimos al espacio en memoria necesario para la manipulación de datos que son creados en **tiempo de ejecución**:
  - **Estructuras dinámicas**: pilas, colas, listas, etc.
  - **Razonamiento**: ¿se puede conocer de ante mano a la ejecución de un programa cuántos elementos tendrá una lista?
- La **memoria dinámica** puede **cambiar** durante la ejecución y será el programador quien deba **asignarla** y **liberarla** explícitamente.

Áreas de memoria

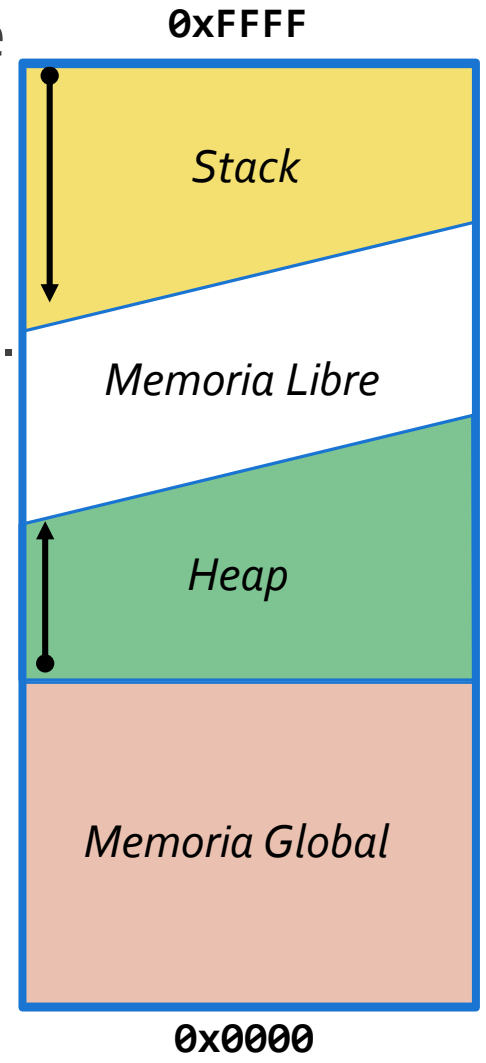
Áreas de Memoria :: esquemático

- Un programa en C almacena sus datos en **tres** áreas diferentes de memoria:
  - Memoria global
  - Stack
  - Heap
- **Memoria global** de tamaño **fijo**.
- **Stack** y **Heap** de tamaño **variable**.



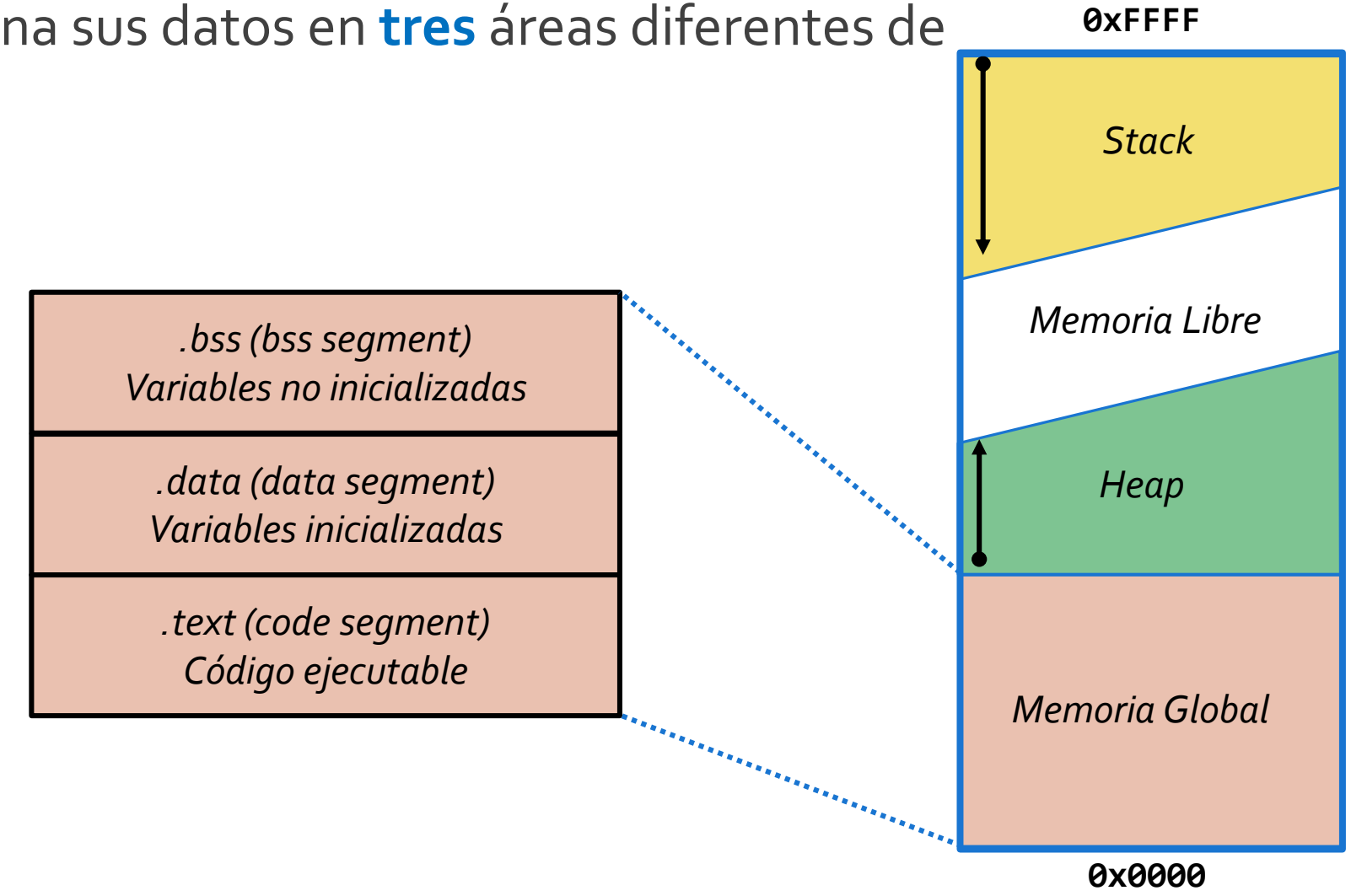
Áreas de Memoria :: esquemático

- Un programa en C almacena sus datos en **tres** áreas diferentes de memoria:
- **Memoria global**
  - Almacena las variables declaradas como **globales** o **estáticas**.
  - Los datos están presentes desde el **comienzo** del programa hasta que este **termina**.
  - El **tamaño** de los datos **no cambia** en ejecución, y es gestionado **automáticamente**.



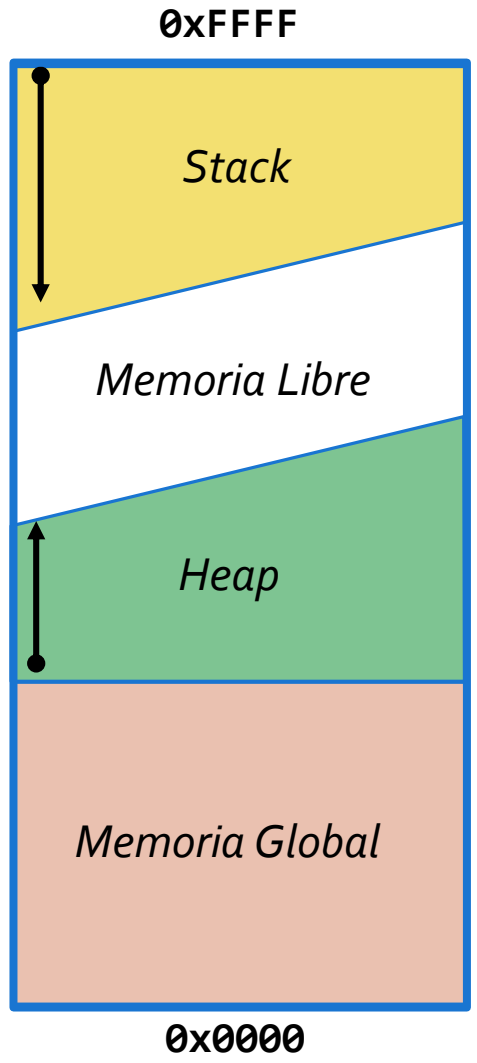
Áreas de Memoria :: esquemático

- Un programa en C almacena sus datos en **tres** áreas diferentes de memoria:
- **Memoria global**



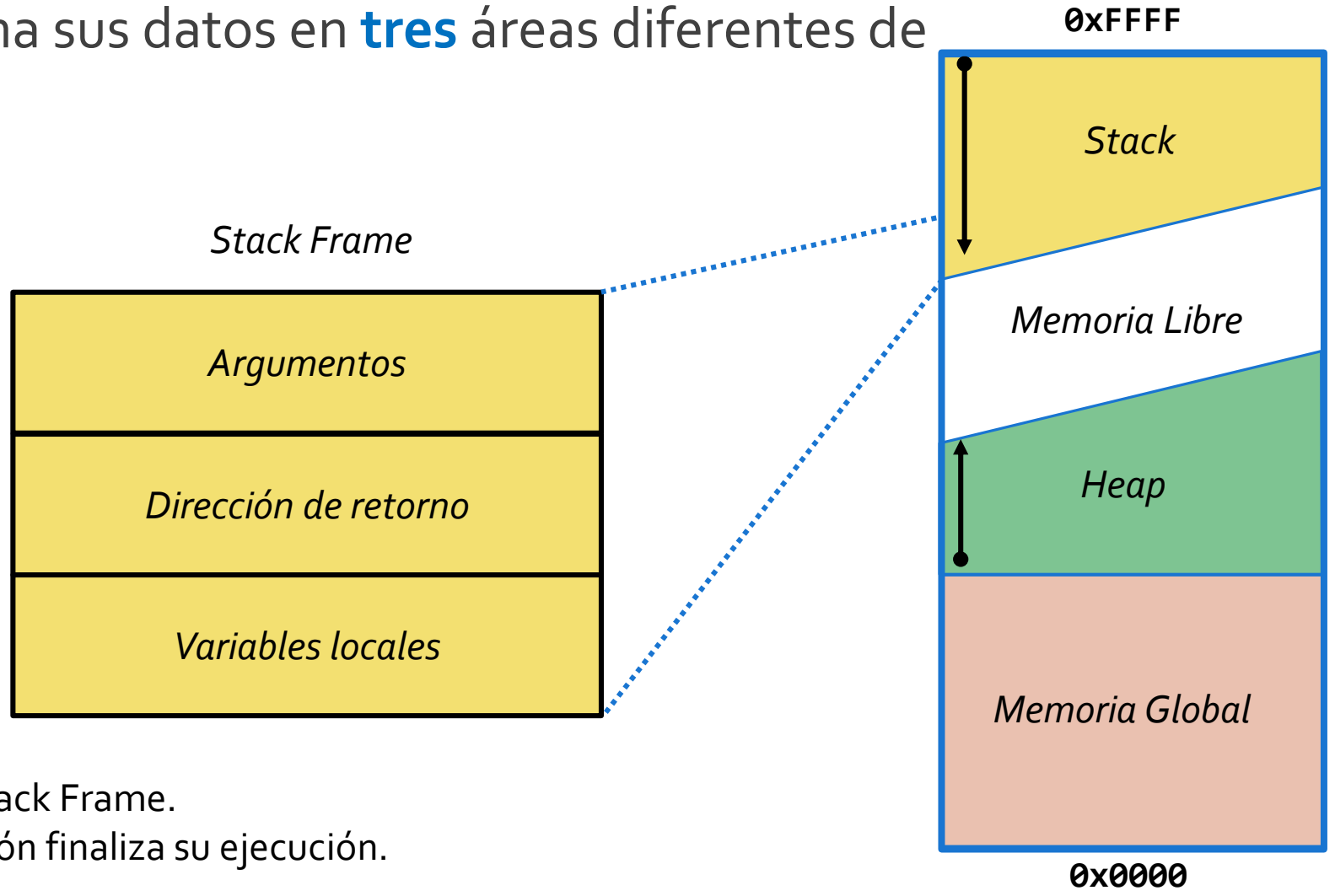
Áreas de Memoria :: esquemático

- Un programa en C almacena sus datos en **tres** áreas diferentes de memoria:
- **Stack**
  - Almacena **datos de control** y variables **locales** a diferentes funciones del programa.
  - Todos los datos almacenados **aparecen** y **desaparecen** en un **momento puntual** de la ejecución.
  - Los datos tienen un **ámbito reducido**, sólo están **disponibles mientras** se ejecuta la función en la que han sido definidas.



Áreas de Memoria :: esquemático

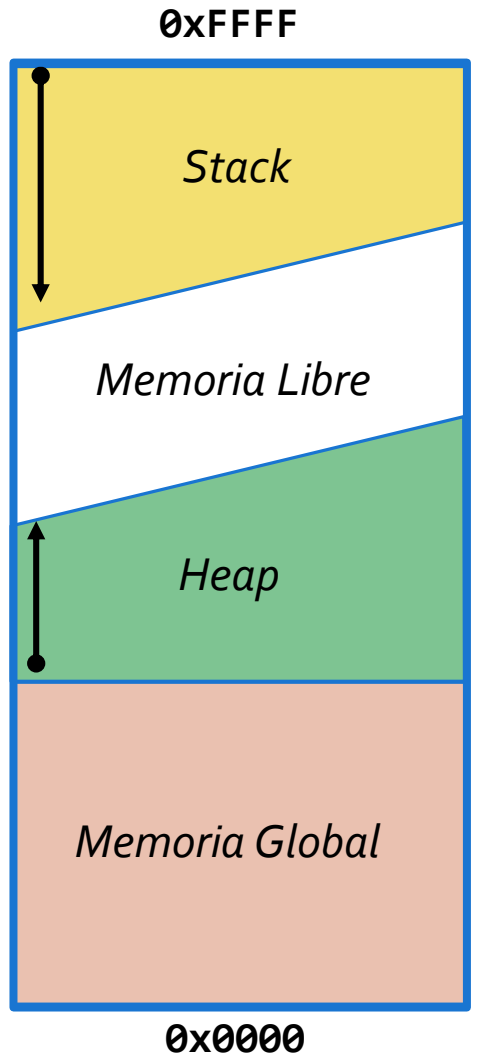
- Un programa en C almacena sus datos en **tres** áreas diferentes de memoria:
- **Stack**



Por cada función invocada se genera un Stack Frame.
Un Stack Frame se elimina cuando la función finaliza su ejecución.

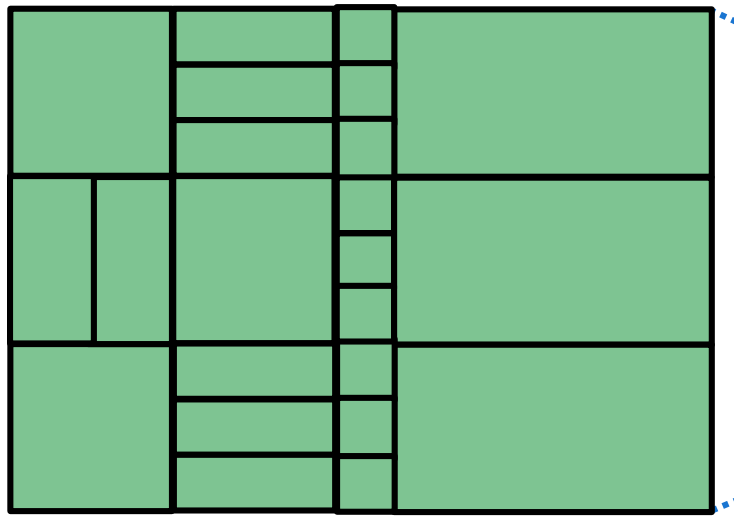
Áreas de Memoria :: esquemático

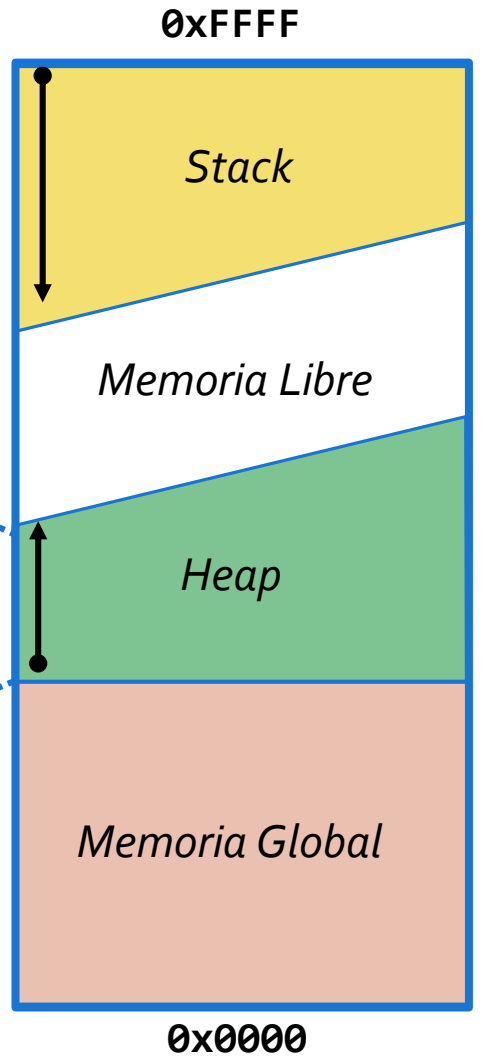
- Un programa en C almacena sus datos en **tres** áreas diferentes de memoria:
- **Heap**
  - **No** almacena datos de variables **globales** o **estáticas**, ni **locales** a las funciones.
  - Es **memoria dinámica** para estructuras de datos con **tamaño desconocido** hasta la ejecución del programa.
  - Contiene memoria **disponible** para que se **reserve** y **libere** en **cualquier** momento durante la ejecución.



Áreas de Memoria :: esquemático

- Un programa en C almacena sus datos en **tres** áreas diferentes de memoria:
- **Heap**





Ante la invocación de **malloc()**, espacio de memoria del heap (montón) es reservado.
El programador es el responsable de liberar dicho espacio con la función **free()** cuando deja de ser requerirlo.

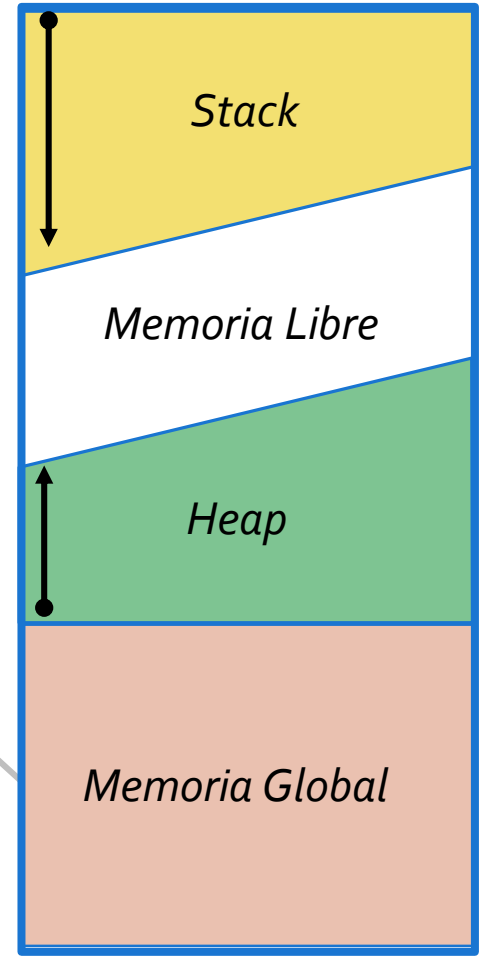
Áreas de memoria :: Ejemplo

Áreas de memoria :: Ejemplo

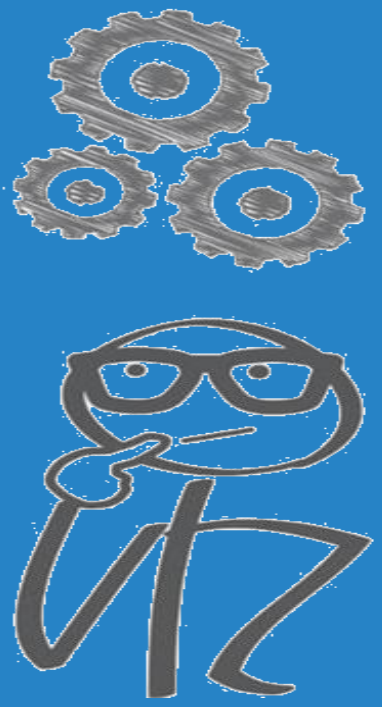
Analizar el siguiente segmento de código.

- ¿Qué datos se alojan en la **memoria global**?

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

struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2;
tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n",
           array[i]->id,
           array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id;
  strcpy(ndata->name, name);
  return ndata;
}
...
```



Áreas de memoria :: Ejemplo



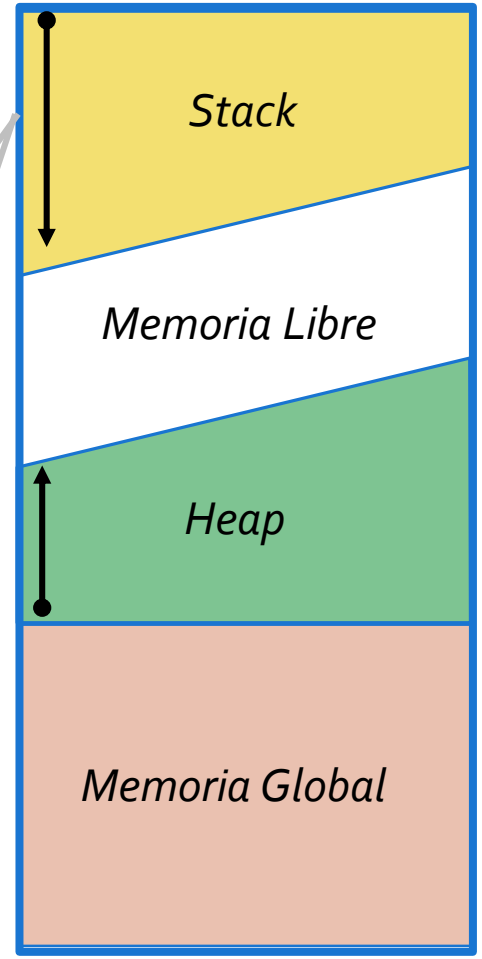
Analizar el siguiente segmento de código.

- ¿Qué datos se alojan en **stack**?

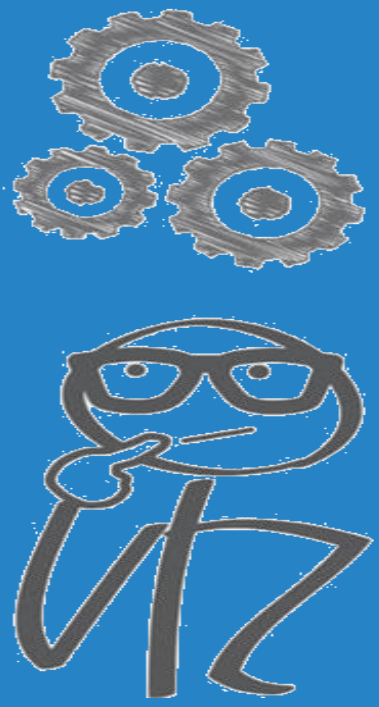
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2;
tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n",
           array[i]->id,
           array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id;
  strcpy(ndata->name, name);
  return ndata;
}
...
```

Cuando se ejecuta

Cuando se ejecuta



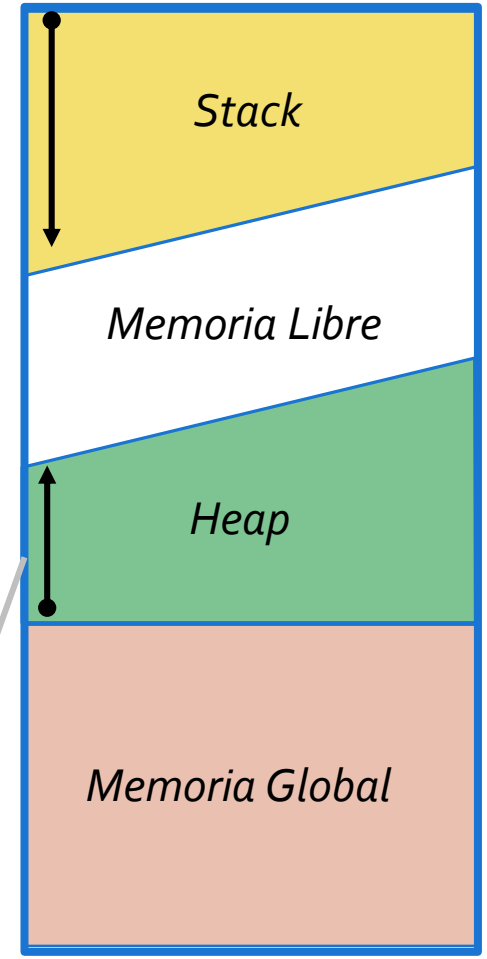
Áreas de memoria :: Ejemplo



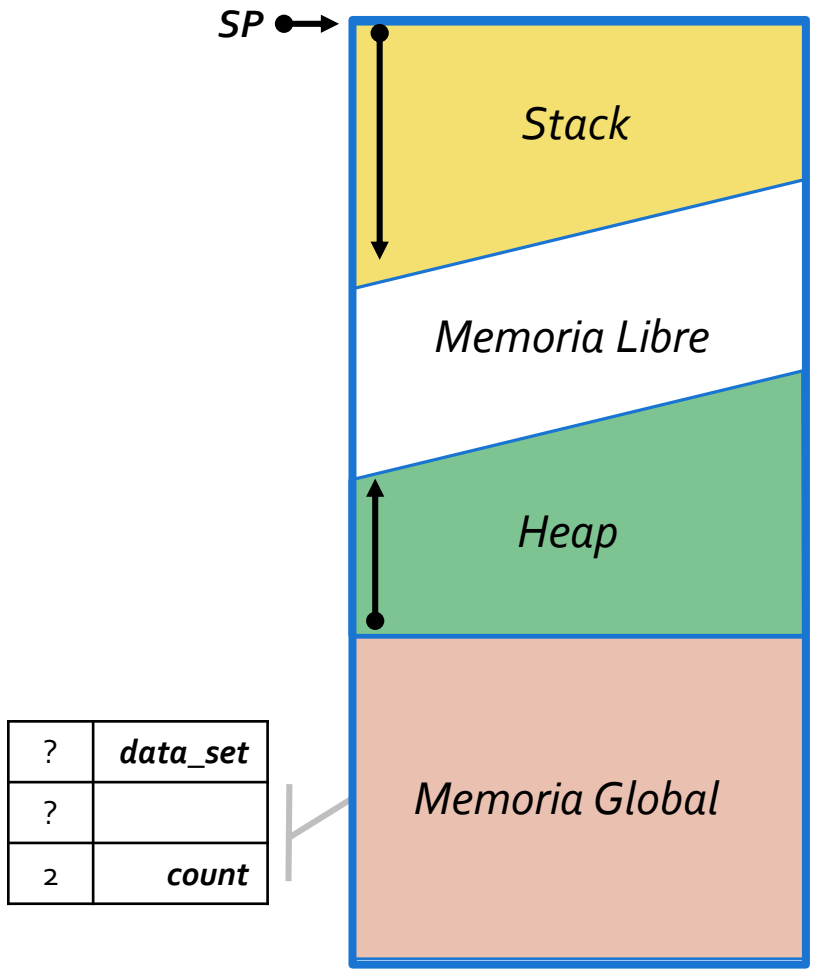
Analizar el siguiente segmento de código.

- ¿Qué datos se alojan en **heap**?

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

struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2;
tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n",
            array[i]->id,
            array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id;
  strcpy(ndata->name, name);
  return ndata;
}
...
```

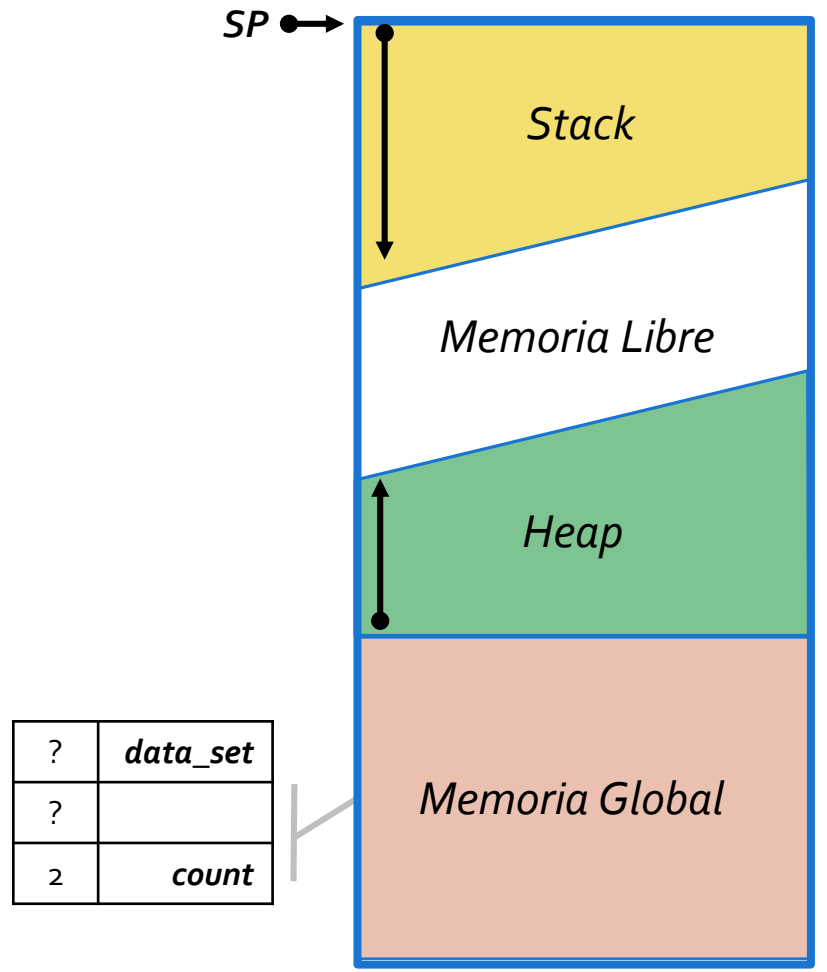


```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



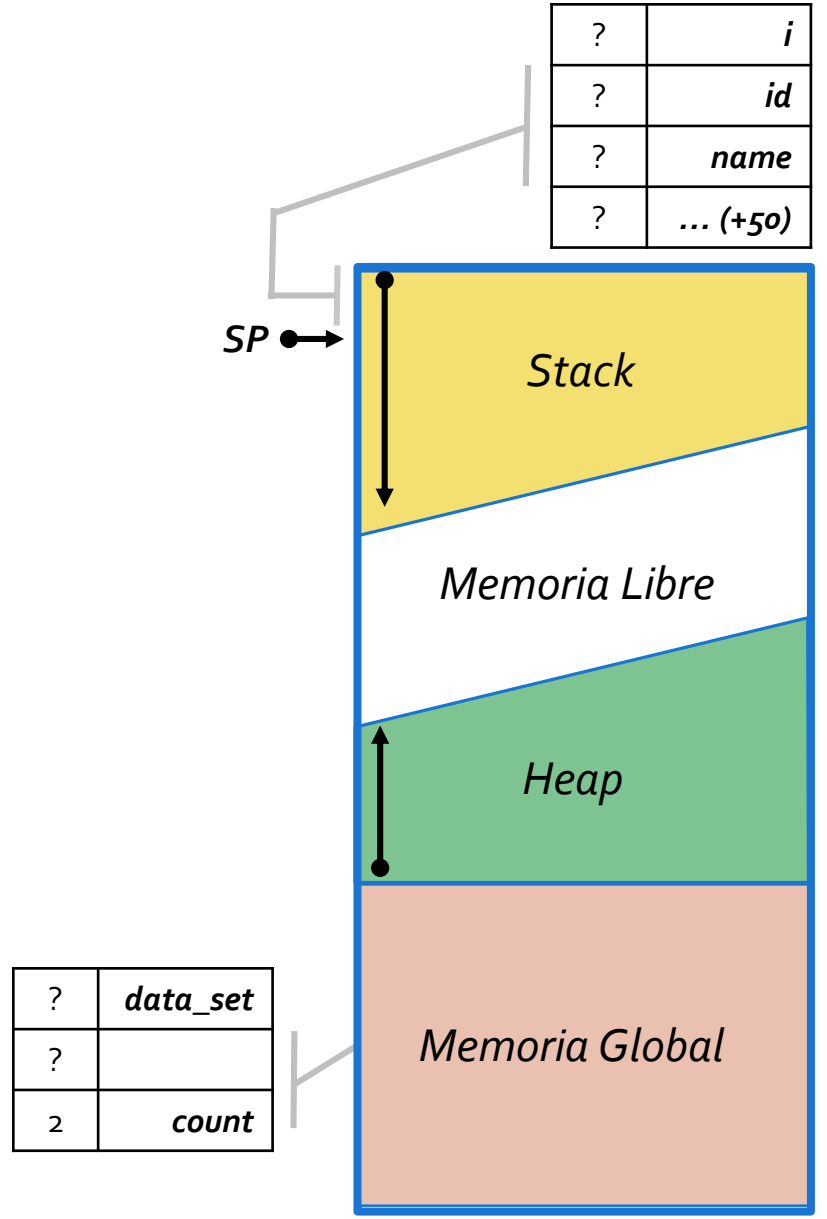
Una vez que el programa es cargado a memoria

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



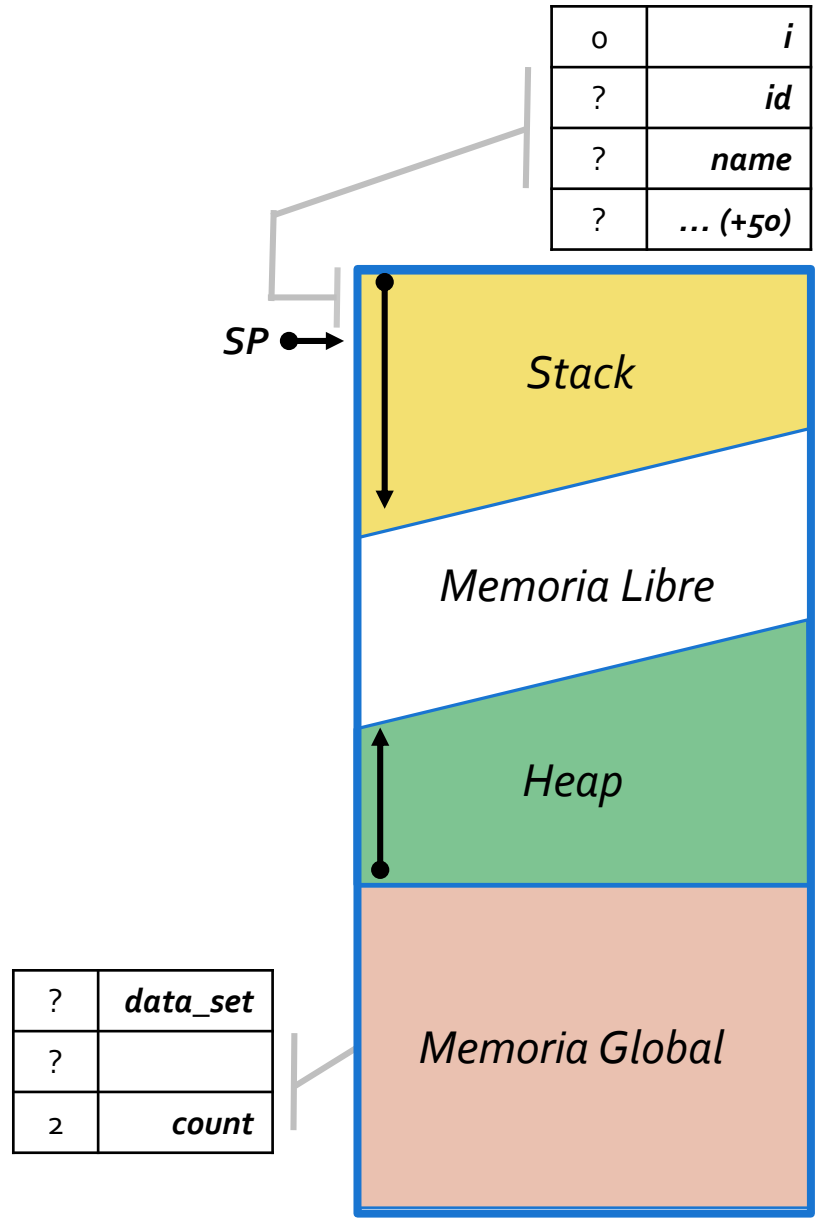
Una vez que se invoca la función main()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



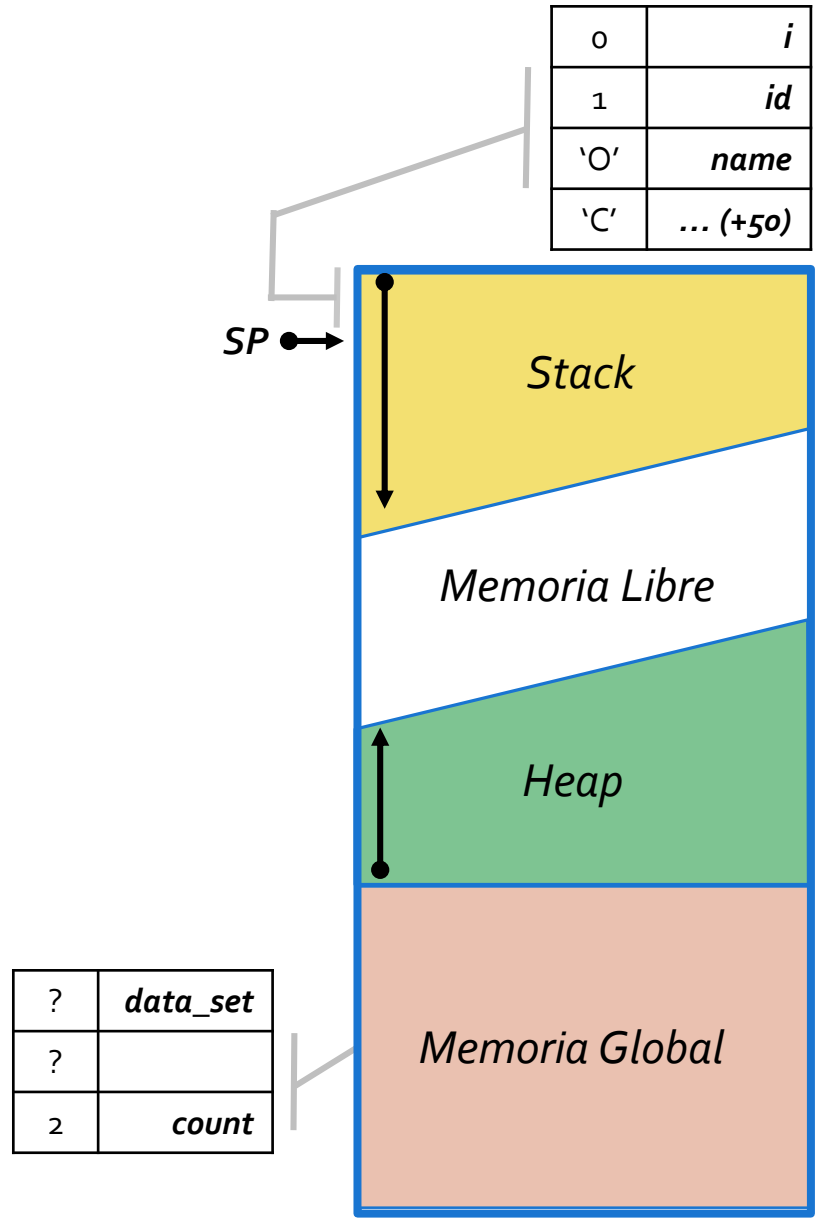
Una vez que se invoca la función main()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



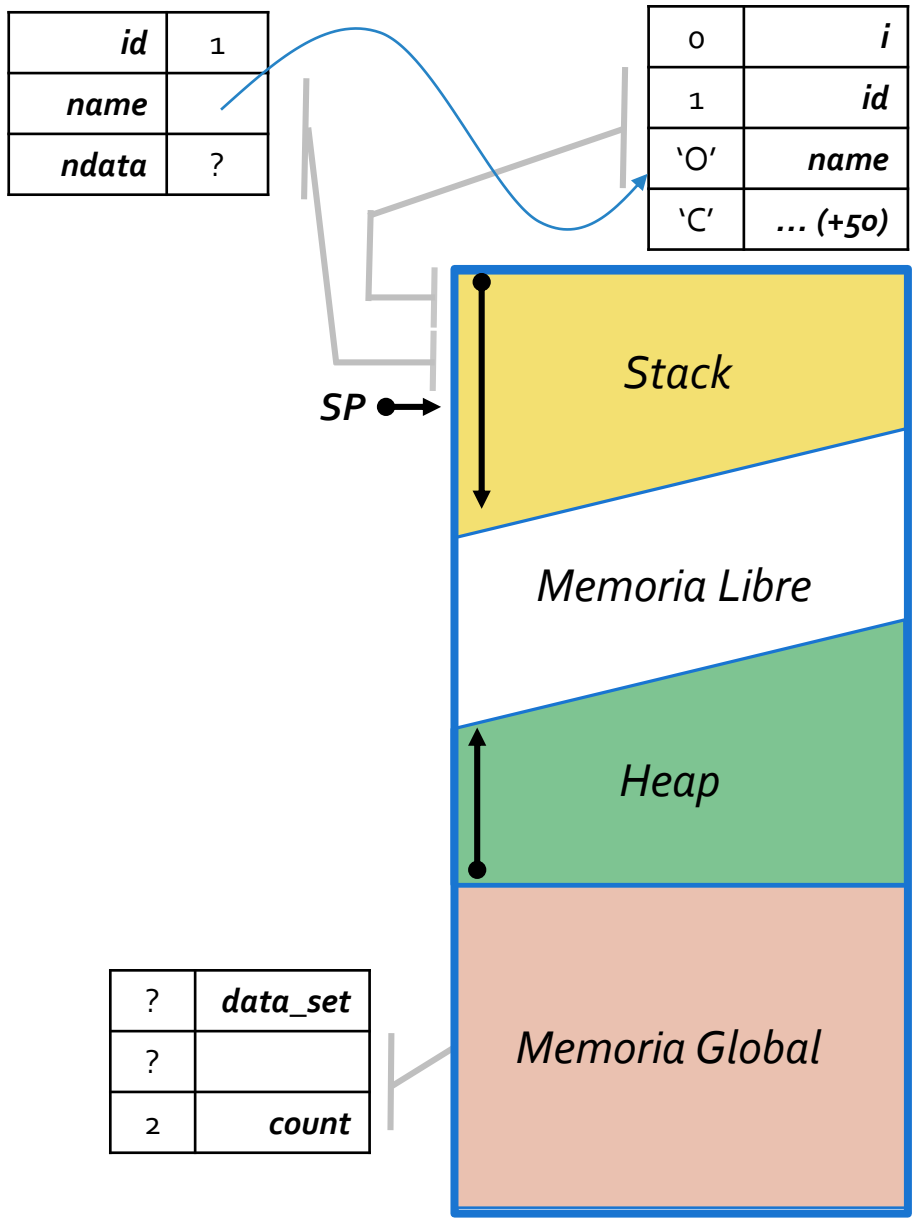
Iteración i=0, se obtienen ID y Nombre

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



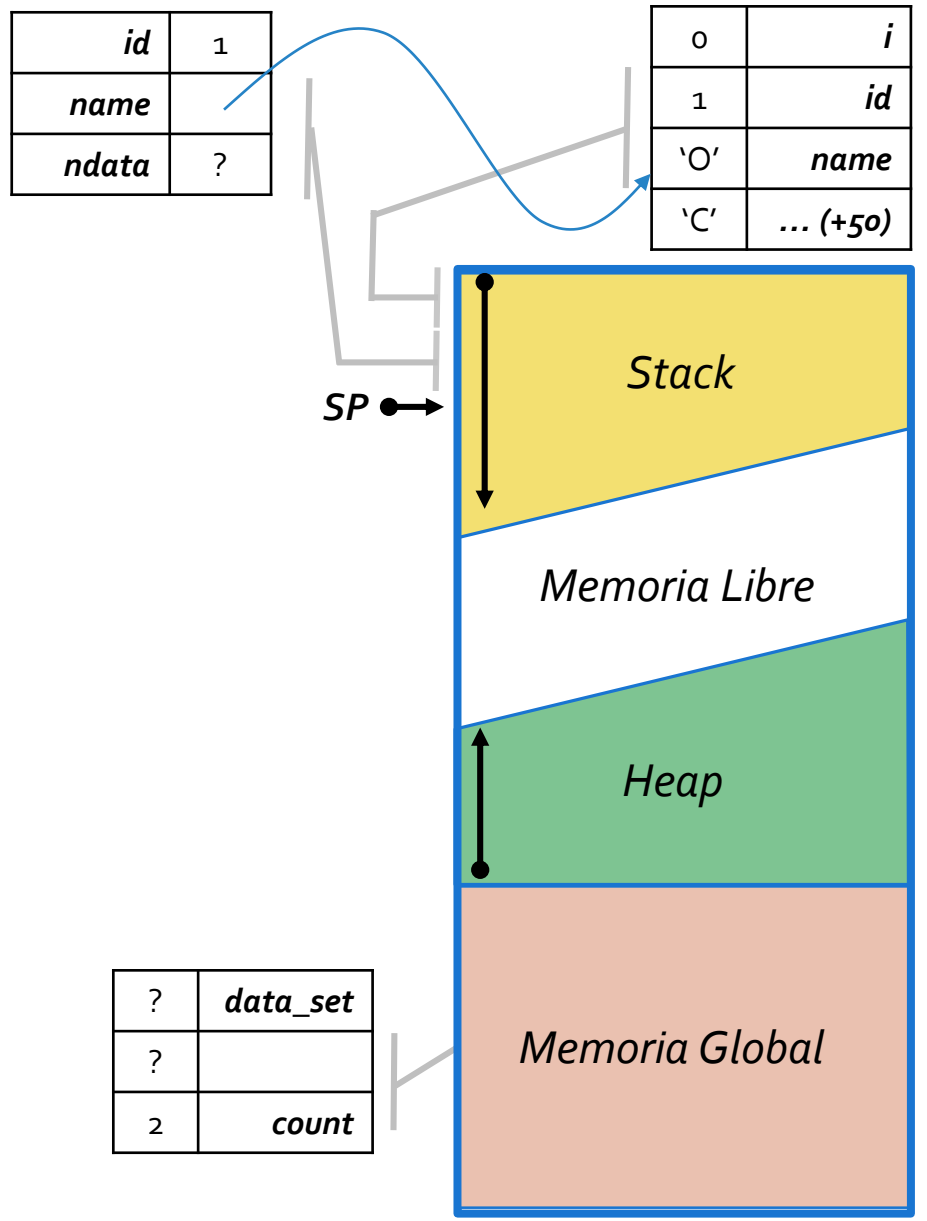
Iteración i=0; se invoca la función new_data()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



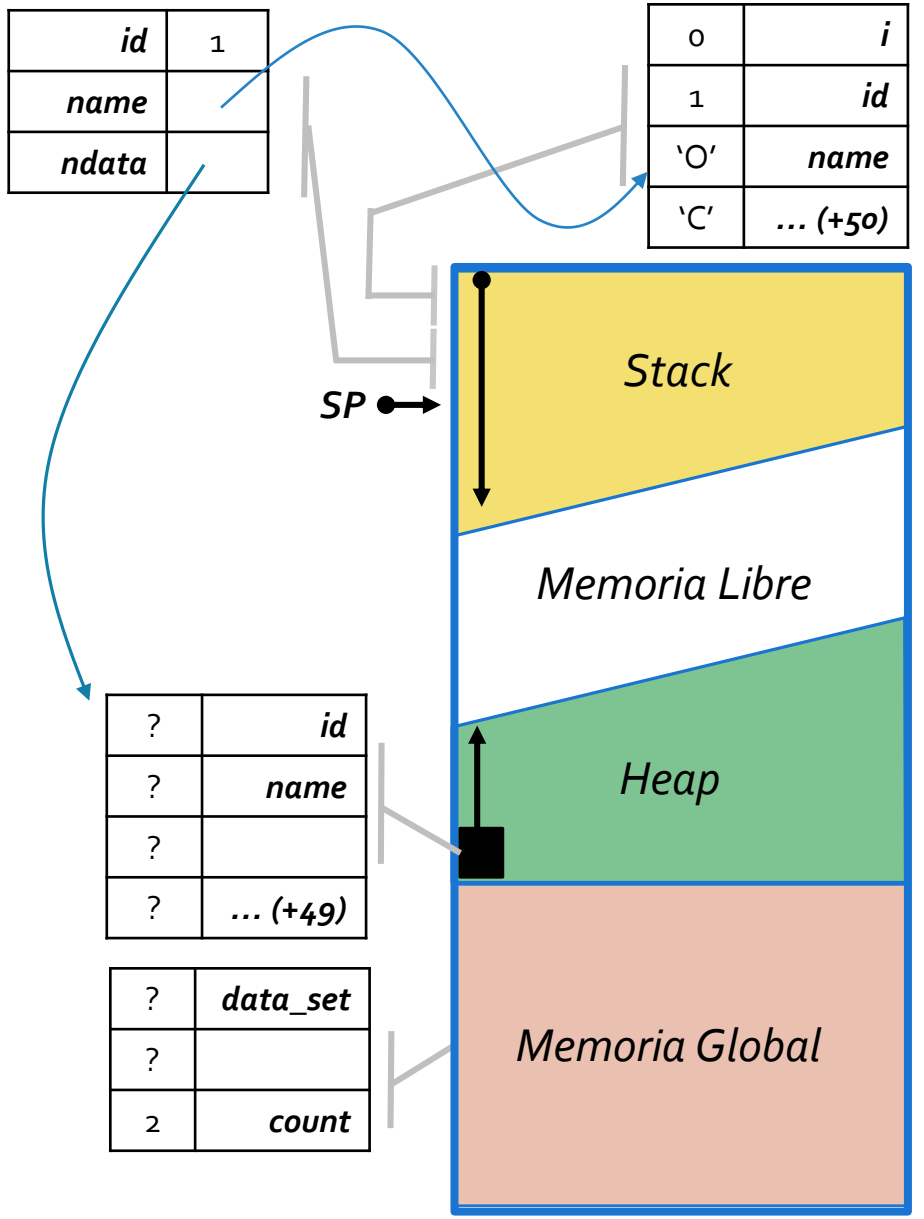
Iteración i=0; se invoca la función new_data()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



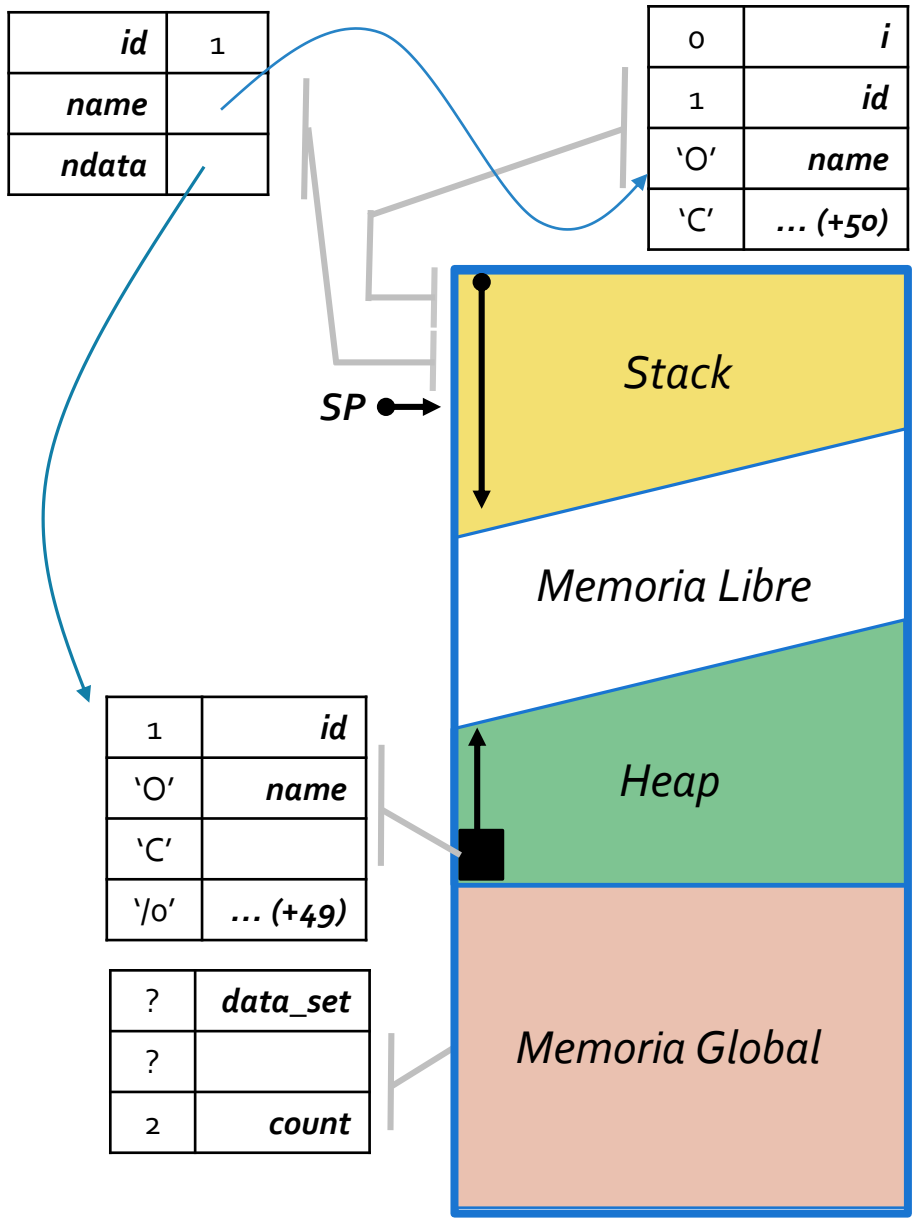
En new_data(), se invoca la función malloc()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



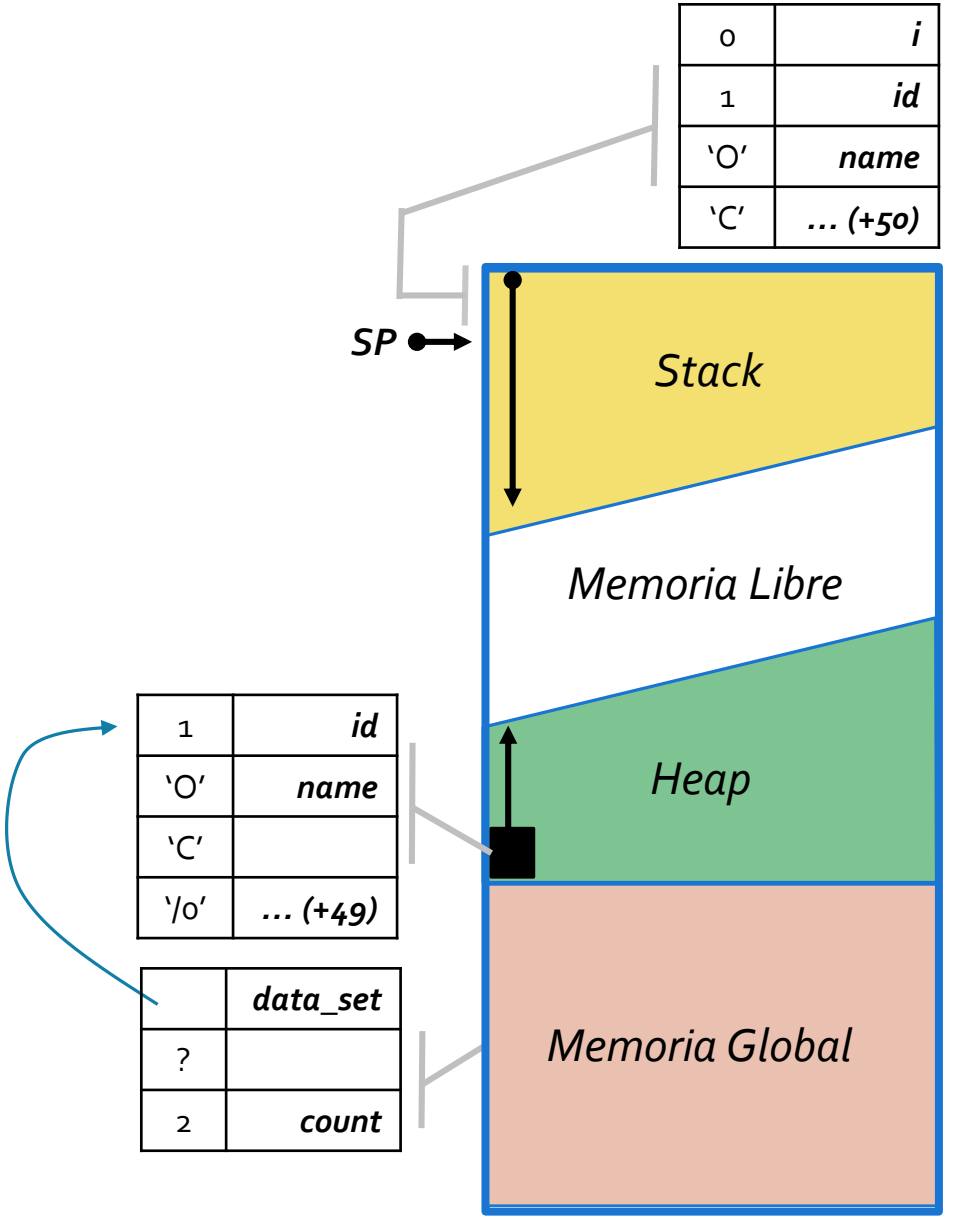
En new_data(), se invoca la función malloc()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



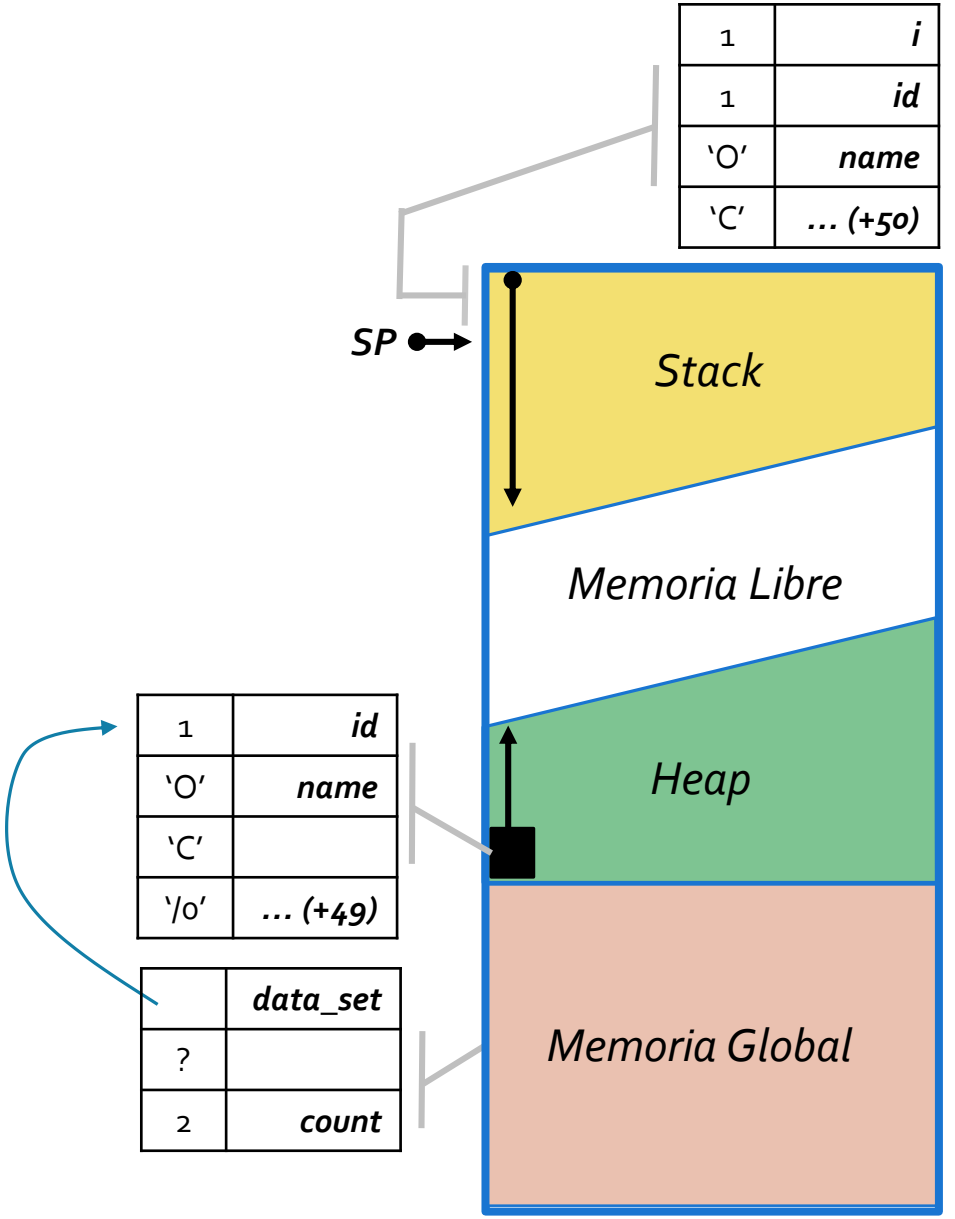
Se copian los datos, finaliza new_data(), y se retorna al main()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



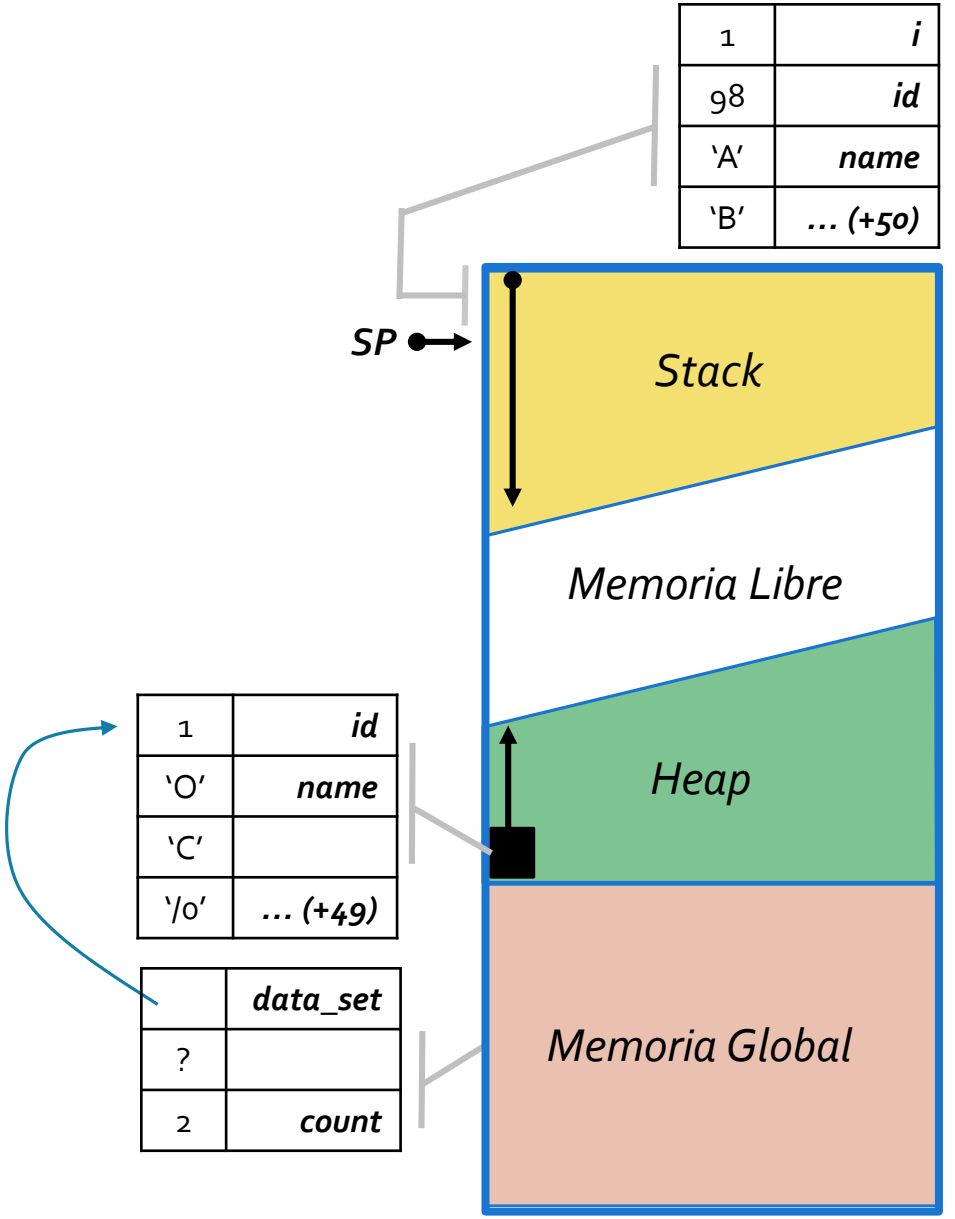
Se copian los datos, finaliza new_data(), y se retorna al main()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



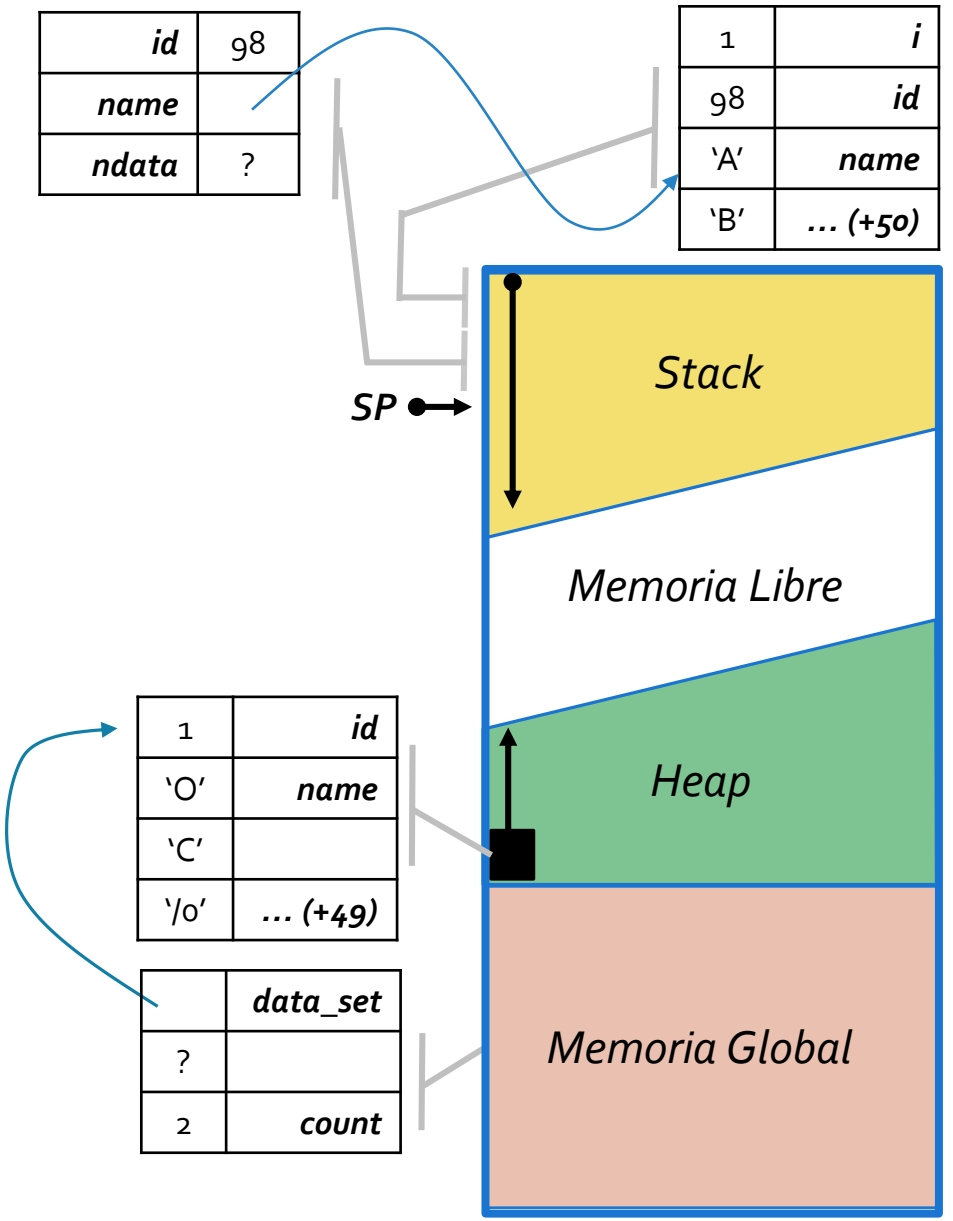
Iteración i=1, se obtienen ID y Nombre

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



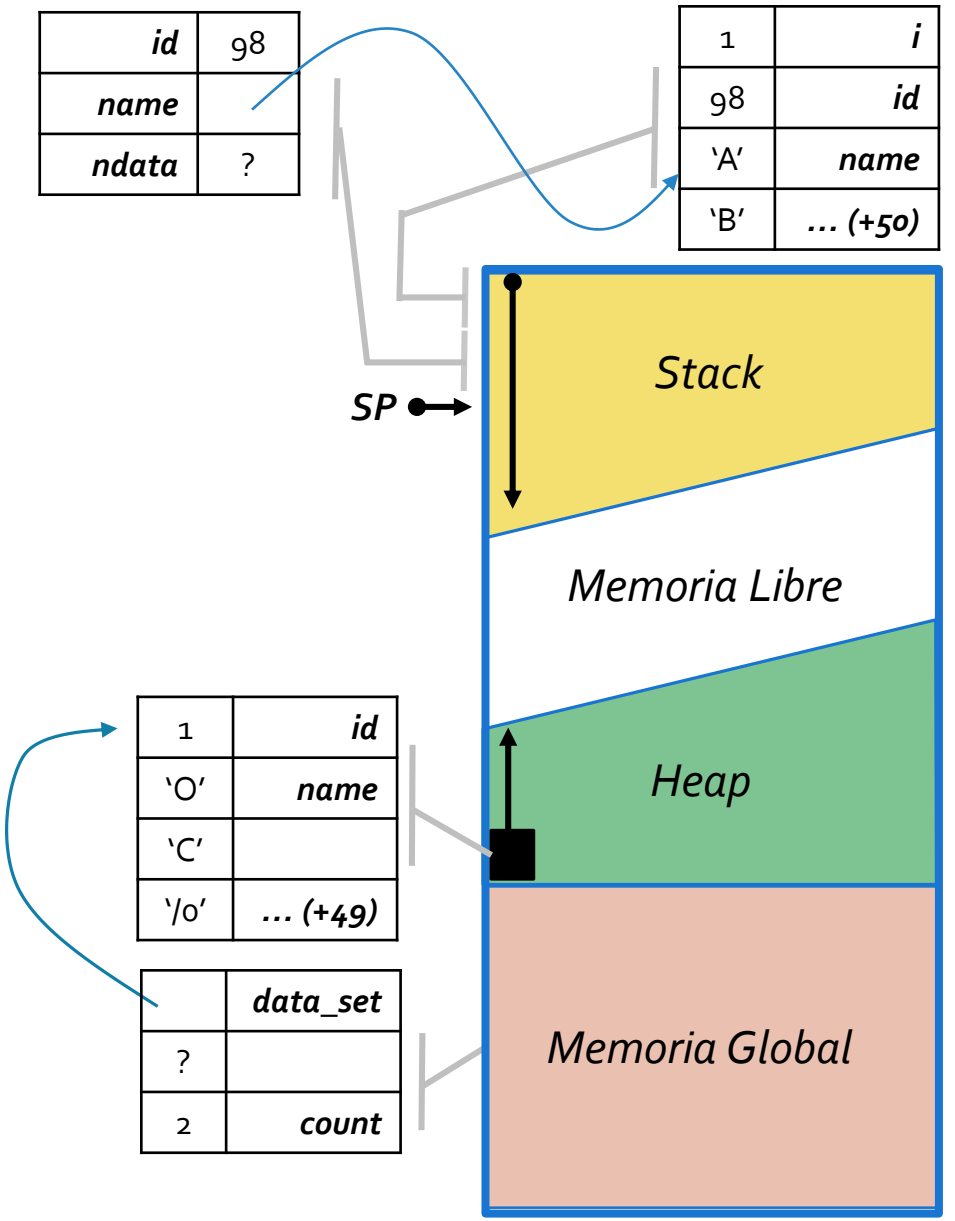
Iteración i=1; se invoca la función new_data()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



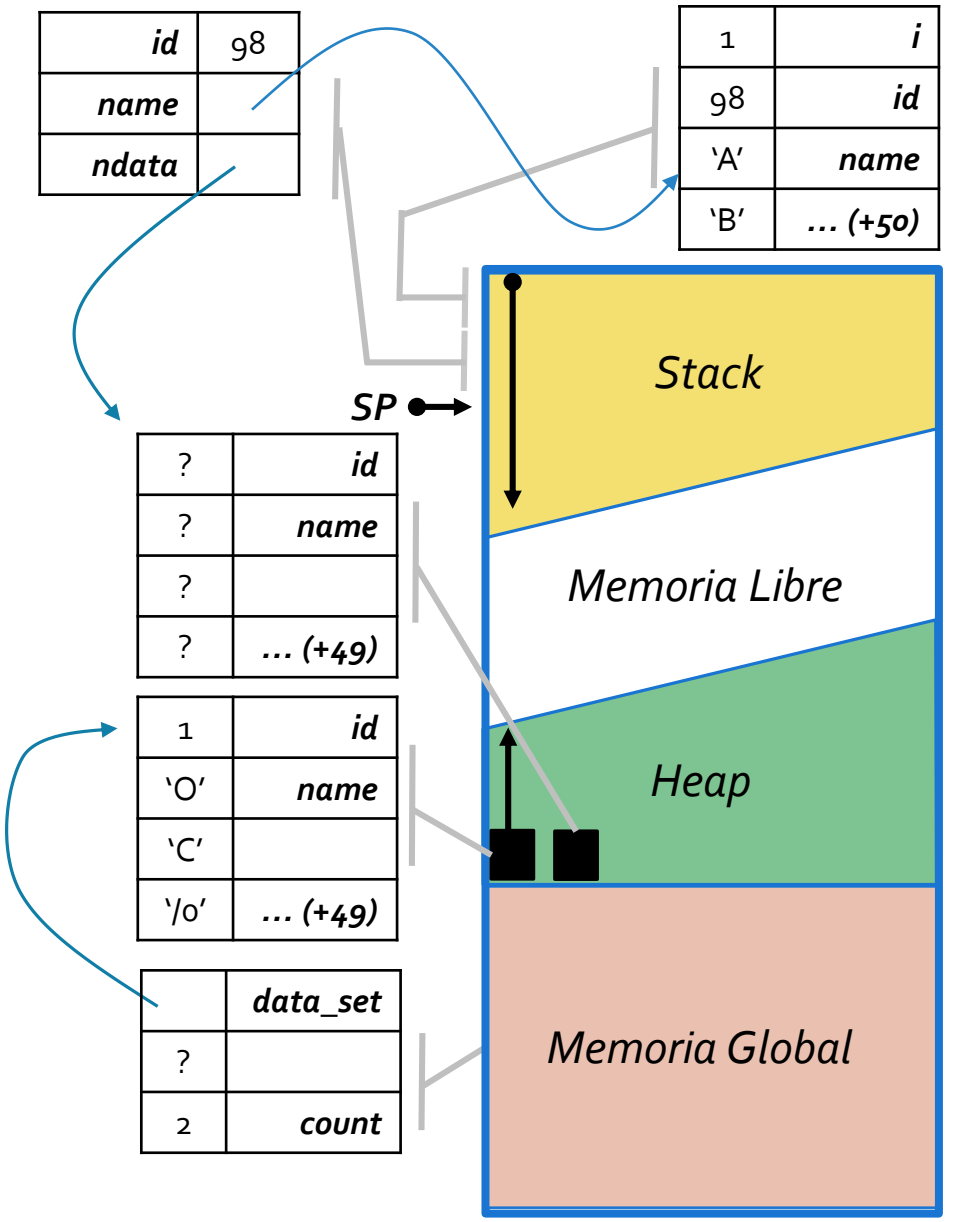
Iteración i=1; se invoca la función new_data()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



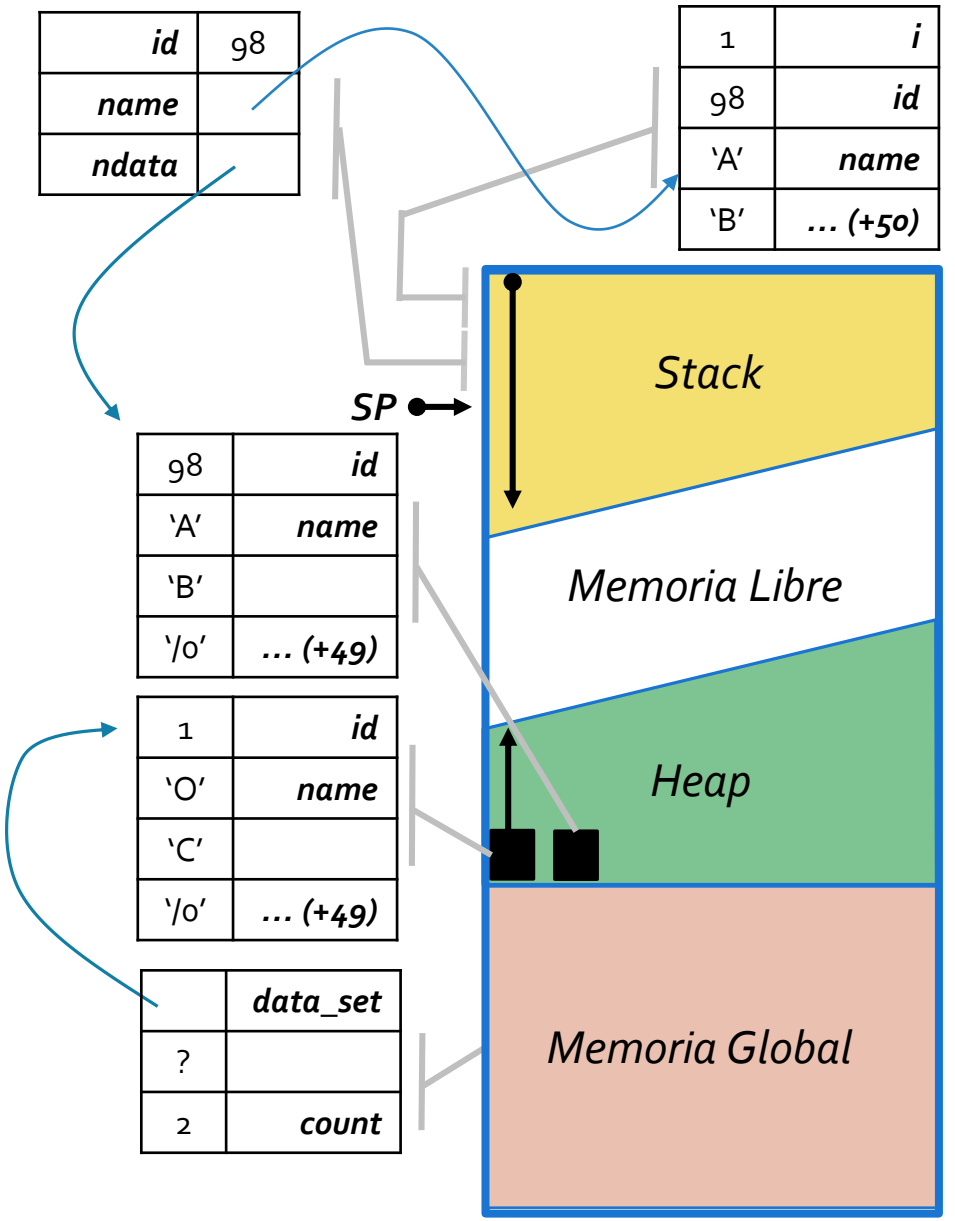
En new_data(), se invoca la función malloc()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



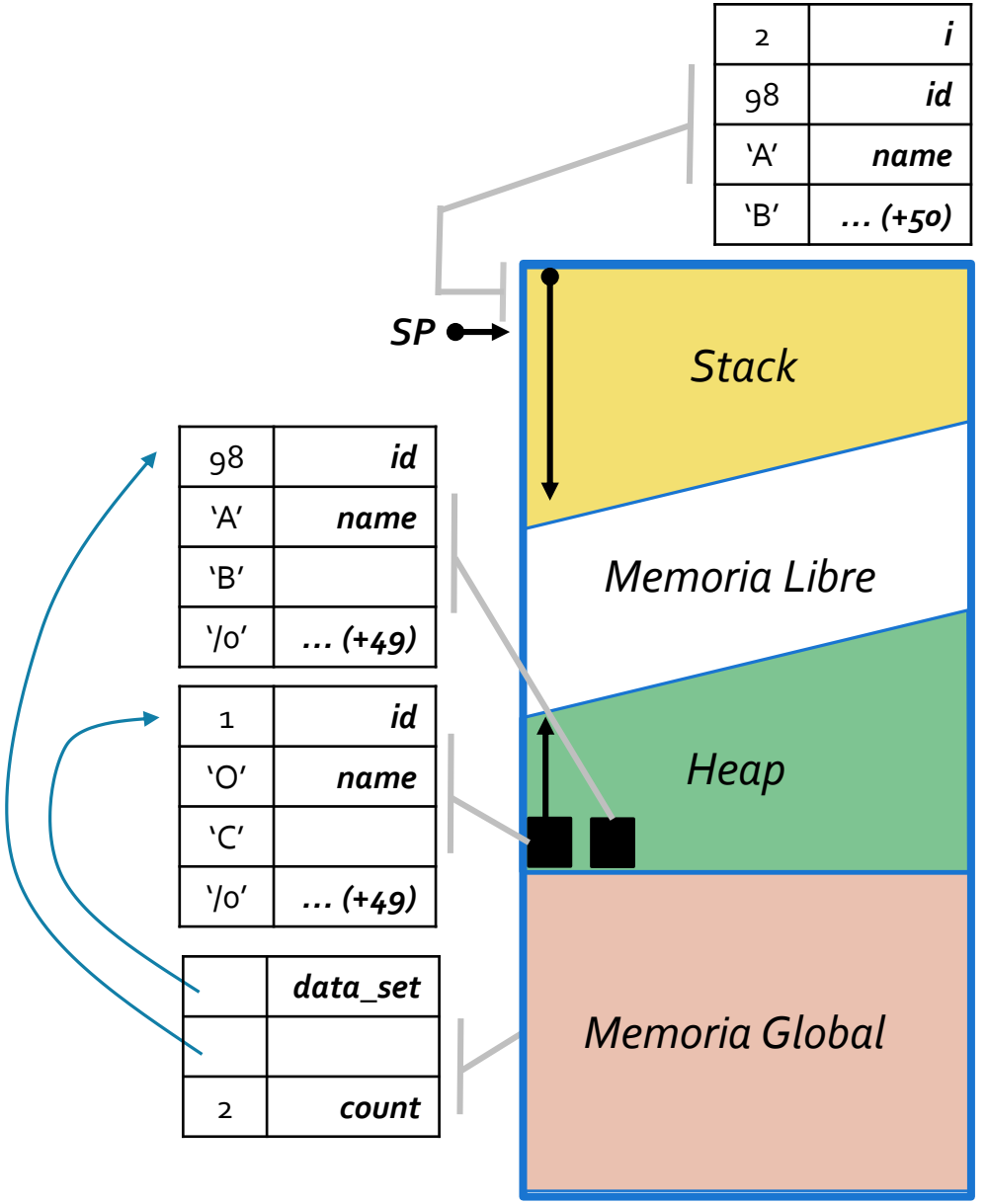
En new_data(), se invoca la función malloc()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



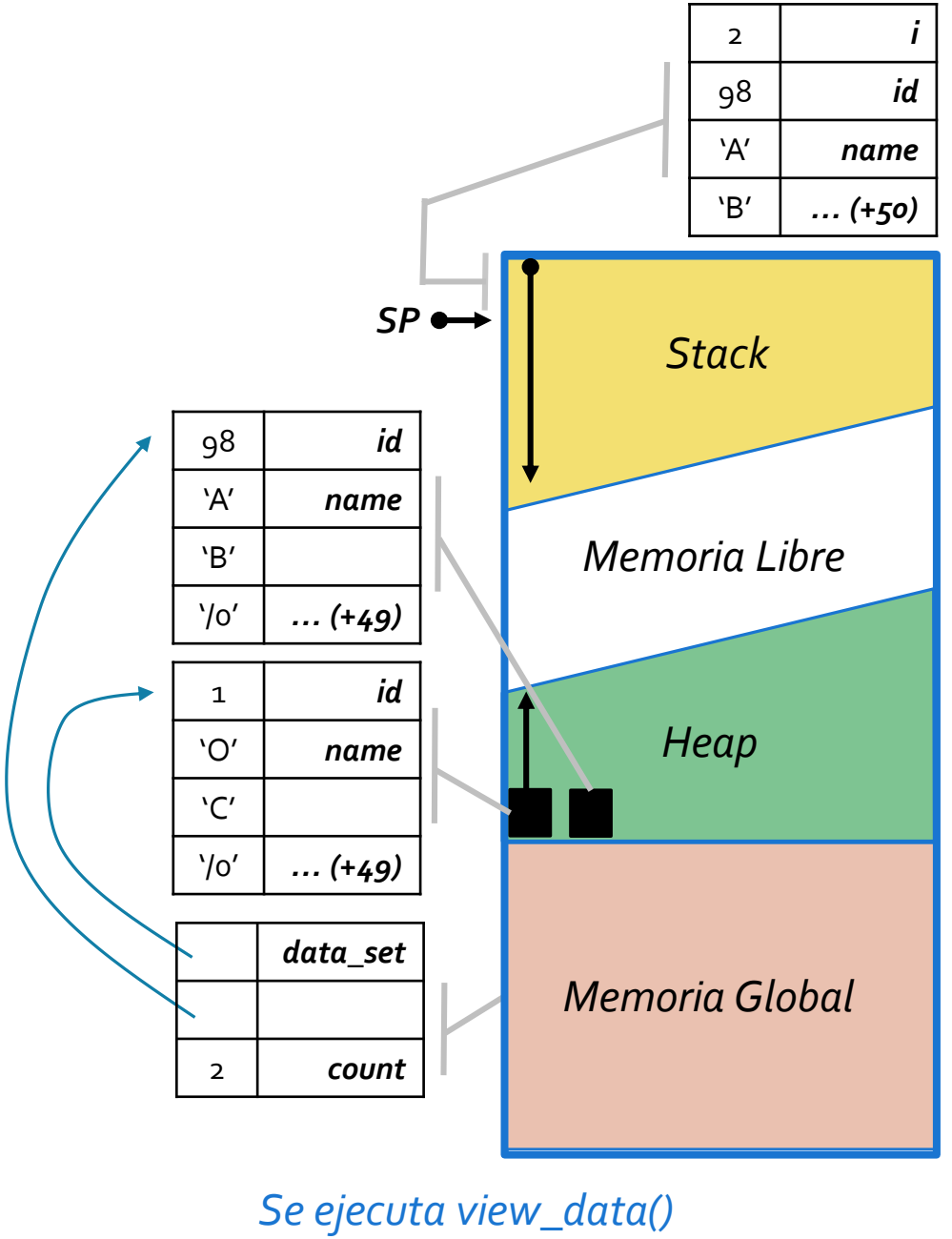
Se copian los datos, finaliza new_data(), y se retorna al main()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



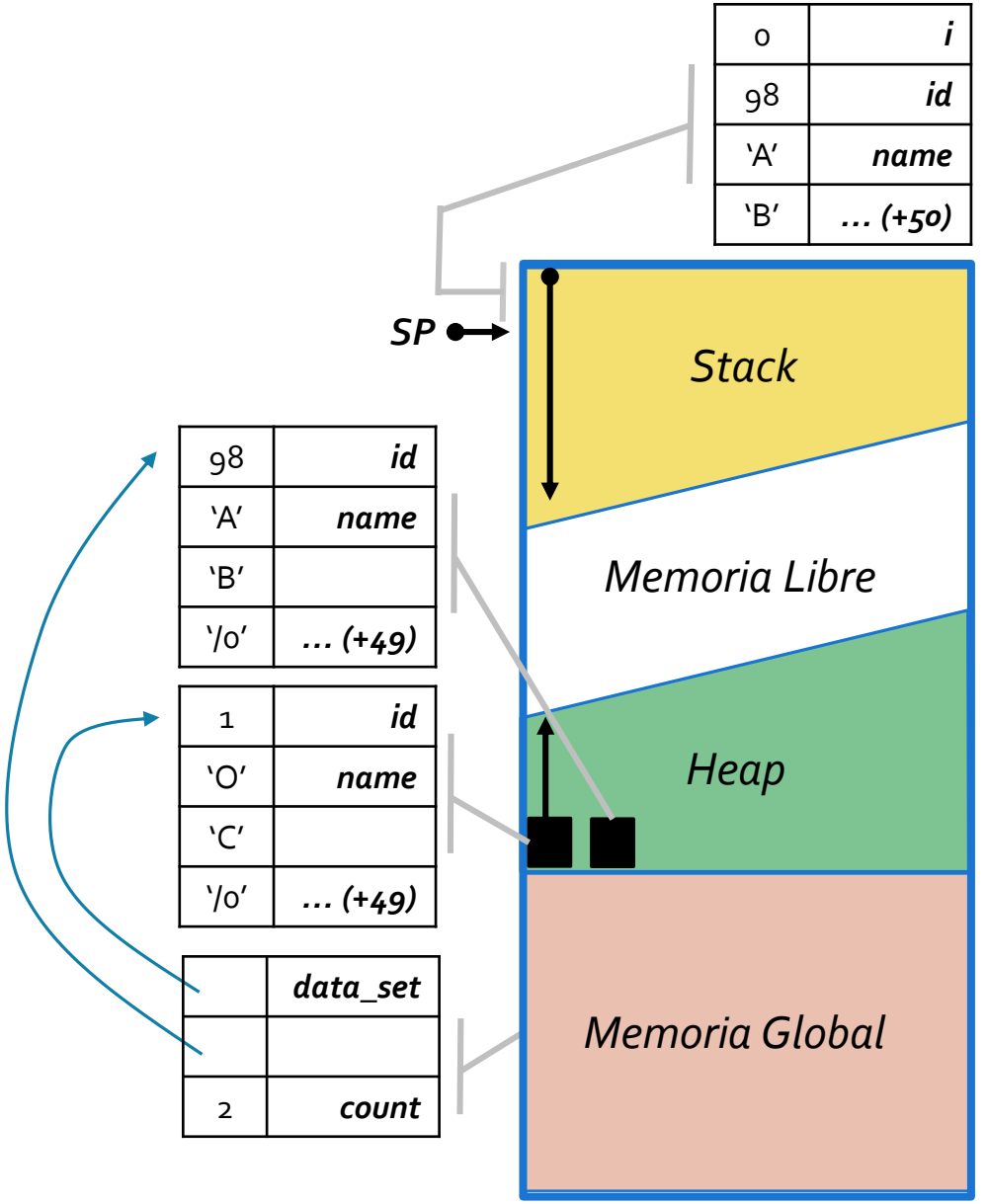
Se copian los datos, finaliza newData(), y se retorna al main()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



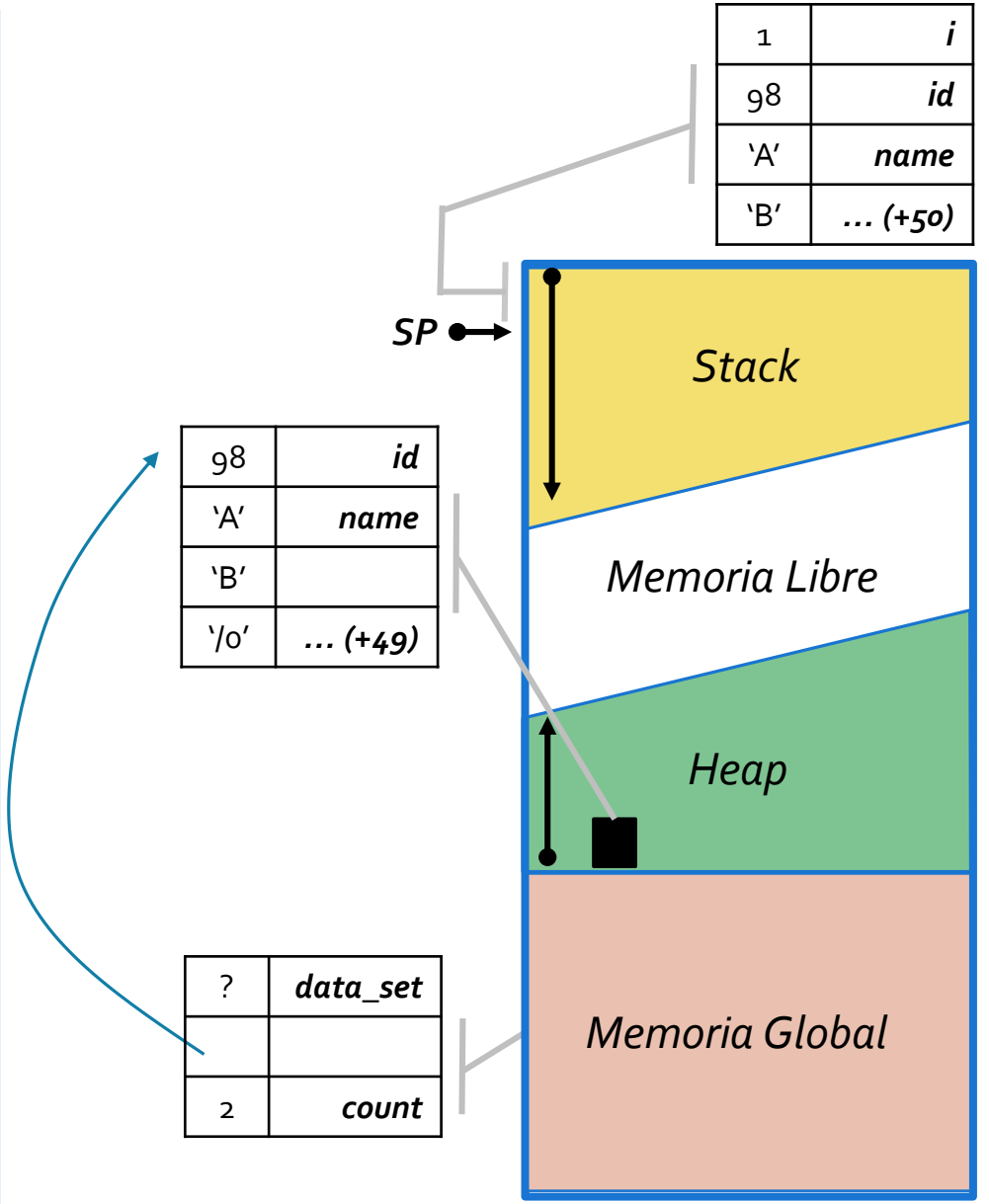
Se ejecuta view_data()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



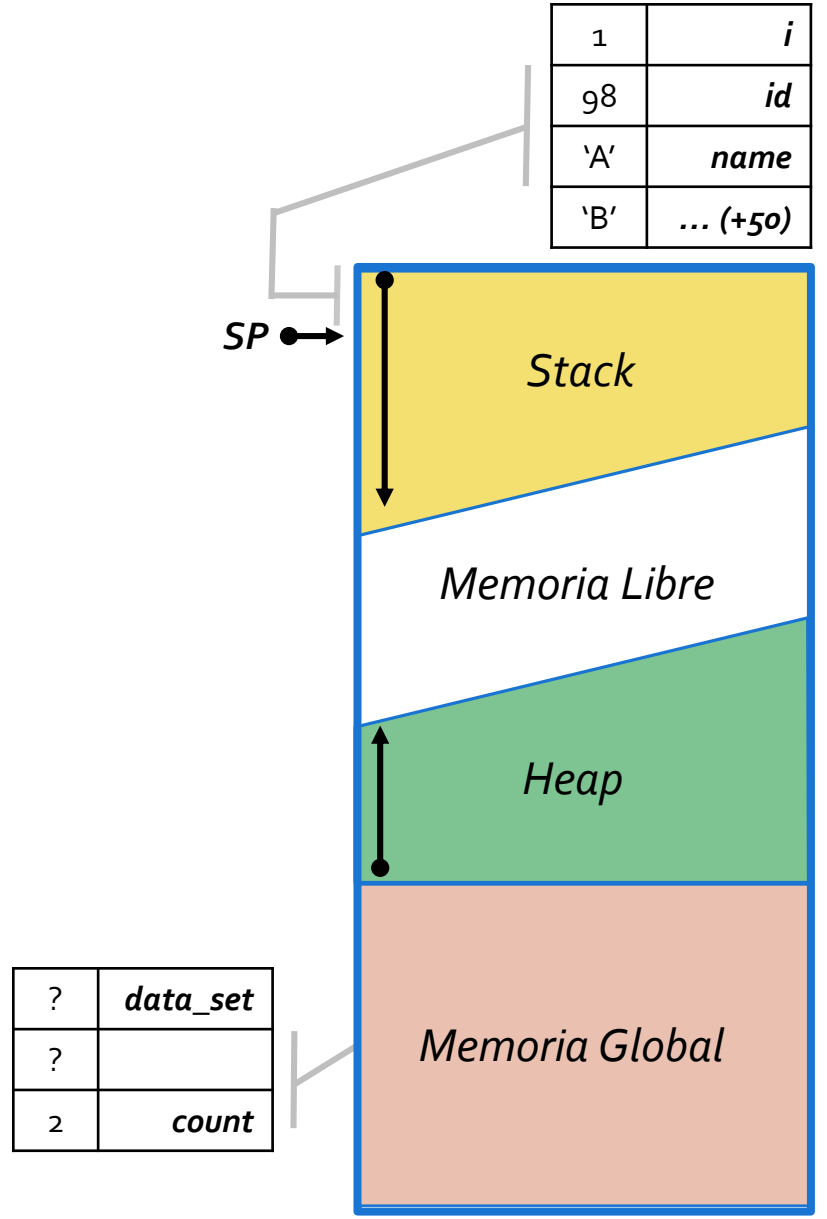
Bucle for, iteración=0, se ejecuta free()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



Bucle for, iteración=1, se ejecuta free()

```c
struct data{ int id; char name [51]; };
typedef struct data * tData;

int count = 2; tData data_set [2];

void view_data(tData * array){
  int i;
  for (i=0; i<count; i++)
    printf("El id: %i corresponde a %s\n", array[i]->id, array[i]->name);
}

tData new_data(int id, char * name){
  tData ndata = (tData) malloc(sizeof(struct data));
  ndata->id = id; strcpy(ndata->name, name);
  return ndata;
}

int main(){
  int i, id; char name [51];

  for(i=0; i<count; i++){
    printf("Introduzca un ID: "); scanf("%d",&id);
    printf("Introduzca un Nombre (max. 50): "); scanf("%50[^\n]", name);
    data_set[i] = new_data(id, name);
  }

  view_data(data_set);
  for(i=0; i< count; i++) { free(data_set[i]); }
  return 0;
}
```



Finaliza el programa, retornando 0.

Administración de memoria :: Errores comunes

Administración de memoria :: Errores comunes

martes, 13 de septiembre de 2022

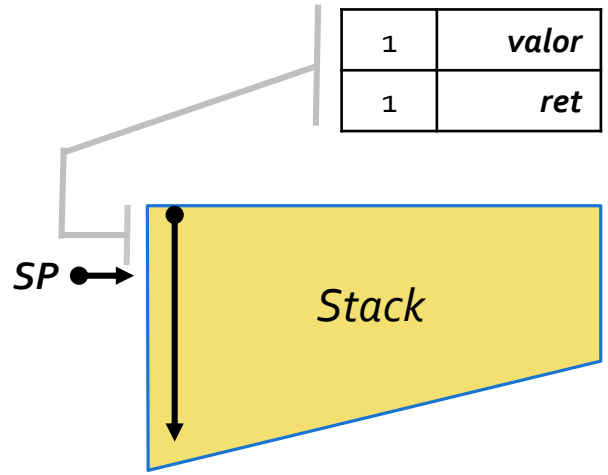
Analizar el siguiente segmento de código.

- ¿Qué es **incorrecto** a la hora de crear un nuevo **int**?

```
int * crear_entero(int valor){
  int ret = valor;
  return &ret;
}
```

warning: function returns address of local variable [-Wreturn-local-addr]

- Se retorna una referencia a la variable local **ret**, que una vez **finalizada** la ejecución de la función **crear_entero**, **deja de persistir** en **Stack**.
- ¿Qué sucede si en esa **locación** de **memoria** dentro del **Stack**, luego, se almacenan datos correspondientes a otra **función**?



Solución correcta:

```
int * crear_entero(int valor){
  int * ret = (int *) malloc(sizeof(int));
  (*ret) = valor;
  return ret;
}
```

Administración de memoria :: Errores comunes

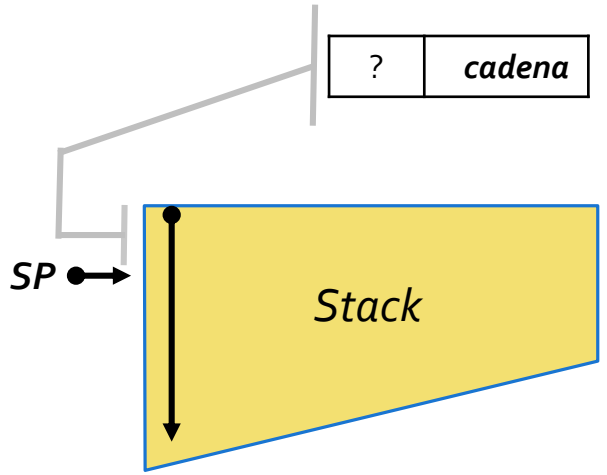
martes, 13 de septiembre de 2022

Analizar el siguiente segmento de código.

- ¿Qué es **incorrecto** cuando se lee una **cadena de char**?

```
void leer_imprimir_cadena(){
  char * cadena;
  scanf("%s", cadena);
  printf("La cadena es: %s\n", cadena);
}
```

warning: 'cadena' is used uninitialized in this function [-Wuninitialized]|



- A la función **scanf** se le parametriza el **puntero a char cadena**, a partir del cual dicha función **escribirá** los datos obtenidos desde la consola.
- ¿**Se reservó espacio** para los **caracteres** que serán **leídos** desde la **consola**?

Solución correcta:

```
void leer_imprimir_cadena(){
  char cadena [101];
  scanf("%100s", cadena);
  printf("La cadena es: %s\n", cadena);
}
```

Administración de memoria :: Errores comunes

martes, 13 de septiembre de 2022

Analizar el siguiente segmento de código.

- ¿Qué es **incorrecto** al crear una nueva **mi_struct**?

```
typedef struct mi_struct{
  int id;
  int flag;
} * t_mi_struct;

t_mi_struct nueva_mi_struct(int i, int f){
  t_mi_struct ret = (t_mi_struct) malloc(sizeof(t_mi_struct));
  ret->id = i;
  ret->flag = f;
  return ret;
}
```

Process terminated with status 0 (0 minute(s), 1 second(s))

*¿Cuál es el tamano de **t_mi_struct** (sizeof(t_mi_struct))?*

Solución correcta:

```
t_mi_struct nueva_mi_struct(int i, int f){
  t_mi_struct ret = (t_mi_struct) malloc(sizeof(struct mi_struct));
  ret->id = i;
  ret->flag = f;
  return ret;
}
```

Fin de la presentación.