Zonas de Memoria

Informática I

R1051

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Hasta ahora, cuando desarrollamos un programa, lo hicimos sin pensar dónde se almacenan los datos utilizados durante la ejecución del mismo. Por ejemplo las variables, sean locales o globales, los datos estáticos o incluso el propio código.

Esto se debe a que el “ordenamiento” de todo lo anterior está controlado por el Sistema Operativo y es, en la mayoría de los casos (o por lo menos a este nivel), transparente al programador.

Sin embargo, es necesario conocer conceptualmente de qué manera se alojan los datos en memoria para comprender, entre otras cosas, el manejo de datos dinámicos y los posibles errores asociados.

Entonces, ¿se almacena todo en el mismo lugar?

```c
#include <stdio.h>

int variable_global = 2;

int suma(int a, int b){
    int suma;
    suma = a + b;
    return suma;
}
int main(int argc, char **argv, char **arge){
    int a = 1;
    int *b = (int *) malloc (sizeof(int));
    int res;
    *b = 2;

    res = suma(a, *b);

    printf("Info 1 + Variables Globales = te comes un %d ;)\n", variable_global);
    free(b);
    return 0;
}

```

Entonces, ¿se almacena todo en el mismo lugar?

```c
#include <stdio.h>

int variable_global = 2;

int suma(int a, int b){
    int suma;
    suma = a + b;
    return suma;
}
int main(int argc, char **argv, char **arge){
    int a = 1;
    int *b = (int *) malloc (sizeof(int));
    int res;
    *b = 2;

    res = suma(a, *b);

    printf("Info 1 + Variables Globales = te comes un %d ;)\n", variable_global);
    free(b);
    return 0;
}

```

Entonces, ¿se almacena todo en el mismo lugar?

```c
#include <stdio.h>

int variable_global = 2;

int suma(int a, int b){
    int suma;
    suma = a + b;
    return suma;
}
int main(int argc, char **argv, char **arge){
    int a = 1;
    int *b = (int *) malloc (sizeof(int));
    int res;
    *b = 2;

    res = suma(a, *b);

    printf("Info 1 + Variables Globales = te comes un %d ;)\n", variable_global);
    free(b);
    return 0;
}

```

¿Cómo sabe, luego de concluir el llamado a “suma” a que dirección de memoria tiene que volver para continuar ejecutando el código donde se produjo el llamado?

Podemos comenzar diferenciando 4 zonas de memoria que serán asignadas por el sistema operativo a nuestro programa con objetivos distintos. Algunas de ellas tienen un tamaño fijo definido en el momento en que se ejecuta el programa y otras varían dependiendo de la sección de código que se esté ejecutando y los requerimientos del mismo. Estas son:

- Zona de Memoria de Código (Code Segment).
- Zona de Datos Estáticos (Data Segment).
- Pila o Call Stack (Stack Segment).
- Zona de Datos Dinámicos o Heap (Heap Segment).

Zona de Memoria de Código:

Es aquí donde se almacena el código propiamente dicho, incluyendo el main y todas las funciones que formen parte de nuestro archivo ejecutable. Una vez configurado todo, se asignará el puntero de instrucciones al comienzo de esta zona (es decir, a la función main) para comenzar la ejecución.

Zona de Datos Estáticos y Globales o simplemente Zona de Datos:

Esta zona de memoria, al igual que la anterior, tiene un tamaño fijo, ya que el espacio necesario para almacenar las “constantes” y las variables globales se conoce desde el comienzo el programa.

Vale la pena aclarar que nada tienen que ver las constantes aquí mencionadas, que representan datos almacenados en memoria cuyo valor no puede ser modificado, con las constantes SIMBÓLICAS asociadas a los “#DEFINE”. Recordemos que estas últimas son simples referencias para el programador, que se reemplazan en tiempo de pre-compilación y a diferencia de los datos estáticos NO OCUPAN MEMORIA.

Pila de llamadas o Call Stack:

Una pila es una estructura de datos donde el último dato almacenado es el primero en extraerse por eso se las denomina LIFO (Last In First Out).

Este sistema se basa fundamentalmente en dos funciones elementales denominadas Push y Pop, que agregan y sacan, respectivamente, un dato del FINAL de la lista (Top of Stack).

La Call Stack, es utilizada para almacenar los datos asociados a los llamados a función, incluyendo los parámetros, las variables locales, la dirección de retorno y una vez finalizada su ejecución, el valor que retorna la función.

Zona de Datos Dinámicos o Heap:

Aquí se encuentran los datos almacenados en espacios de memoria pedida en tiempo de ejecución (utilizando la función *malloc( )*). Como ya hemos visto, esta memoria debe devolverse utilizando la función *free( )*.

Estos dos Segmentos de memoria tienen un tamaño variable.

Resumen

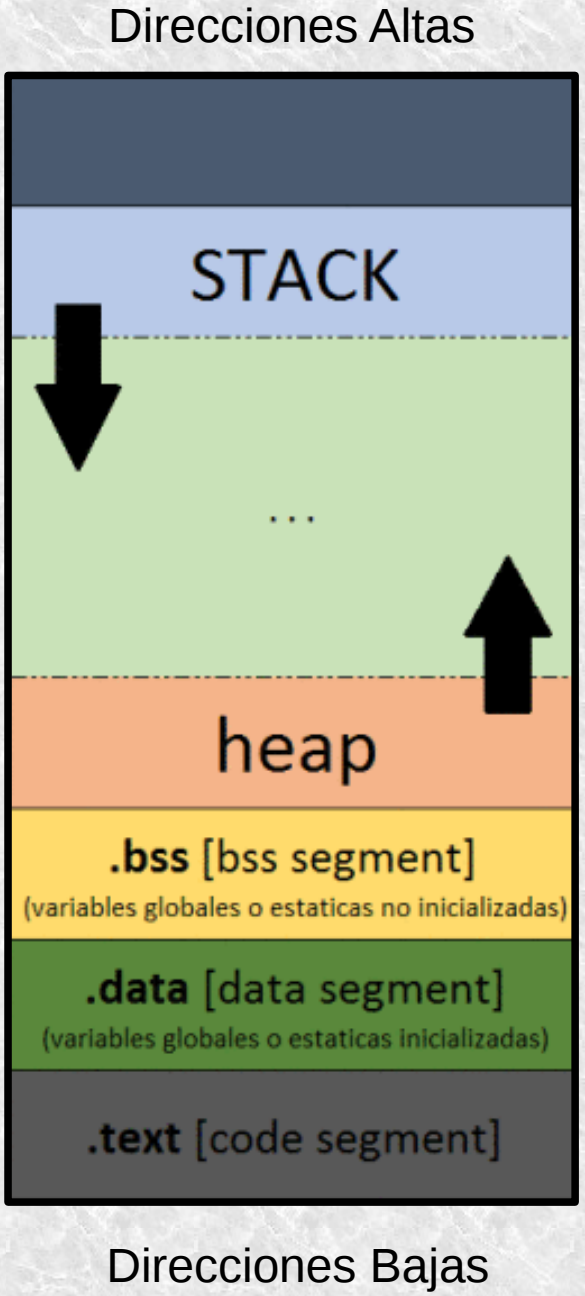
Argumentos de línea de comandos y variables de entorno

Zona de memoria asociada a los datos utilizados en los llamados a función. Crece “hacia abajo”

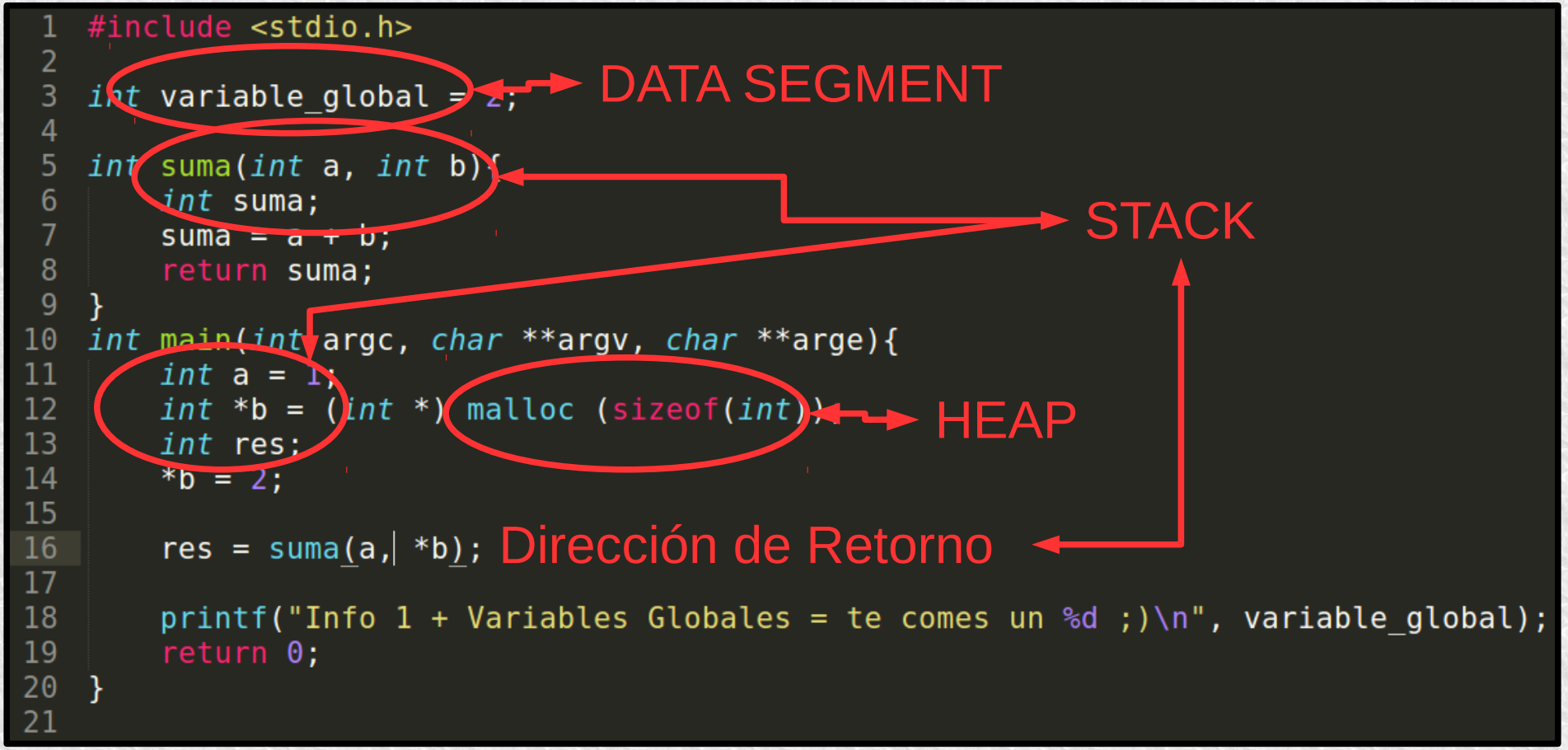
Zona de memoria reservada dinámicamente con la función *malloc( )*. Crece “hacia arriba”

Zona de memoria de datos estáticos y globales. Su tamaño se define al comienzo de la ejecución y permanece invariante.

Zona de Código. Se almacenan las instrucciones que componen nuestro programa



Volviendo...




```c
#include <stdio.h>

int variable_global = 2;

int suma(int a, int b){
    int suma;
    suma = a + b;
    return suma;
}
int main(int argc, char **argv, char **arge){
    int a = 1;
    int *b = (int *) malloc (sizeof(int));
    int res;
    *b = 2;

    res = suma(a, *b);

    printf("Info 1 + Variables Globales = te comes un %d ;)\n", variable_global);
    return 0;
}

```

Gráficamente