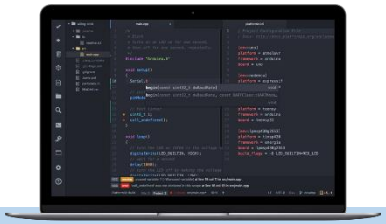
Tema 7

- Memoria dinámica.
- Stack
- Heap
- Funciones para manejo del heap

Tiempo de vida una aplicación



El desarrollador escribe el código del la aplicación, diseña las estructuras de datos, los algoritmos.

En c, el tiempo de compilación se lleva a cabo solamente una vez, traduciendo el código fuente a un archivo ejecutable.

Se lee a partir de la traducción del código fuente y se llevar a cabo las tareas que el programador ha incorporado en el código.

Tiempo de diseño

Tiempo de compilación

Alguno de los errores que se pueden producir se encuentran:

- Errores de sintaxis, como por ejemplo: Falta un “;” faltan símbolos de cierre.
- Falta de alguna librería

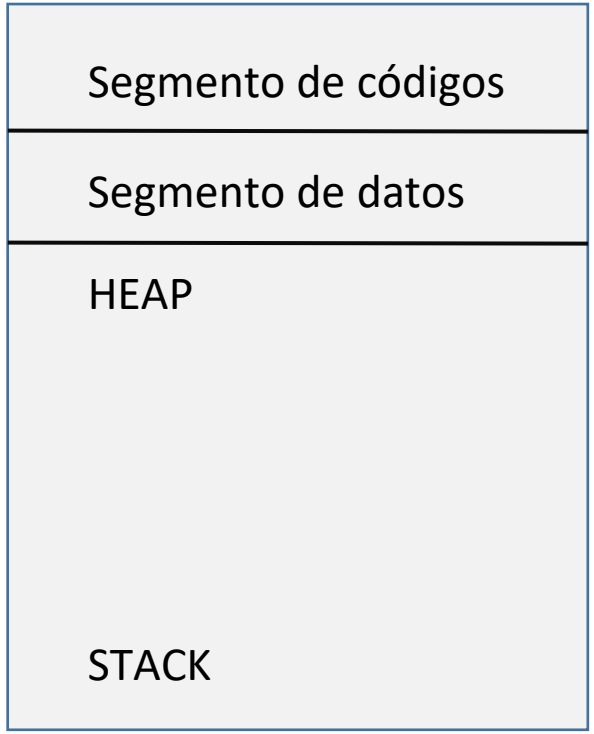
Tiempo de Ejecución

Alguno de los errores que se pueden producir se encuentran:

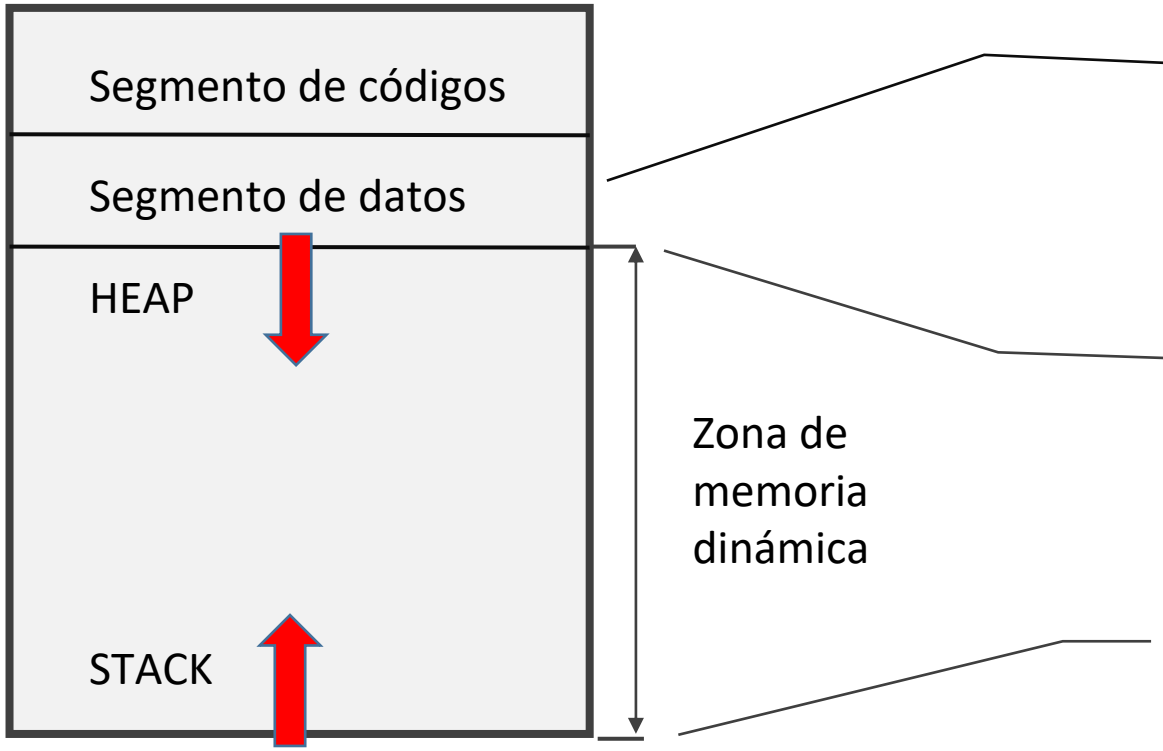
- División entre cero.
- Asignación forzadas de tipos.
- Acceso a memoria restringida

Llamada a una aplicación

SE EJECUTA UN PROGRAMA



Llamada a una aplicación



El Segmento de Datos contendrá aquellos identificadores declarados en **forma global (fuera de toda función)**

Heap, sólo se ocupa cuando se solicitan **reservas dinámicas**, o sea durante la **ejecución** y recién entonces se decide **cuántos bytes** se requieren.

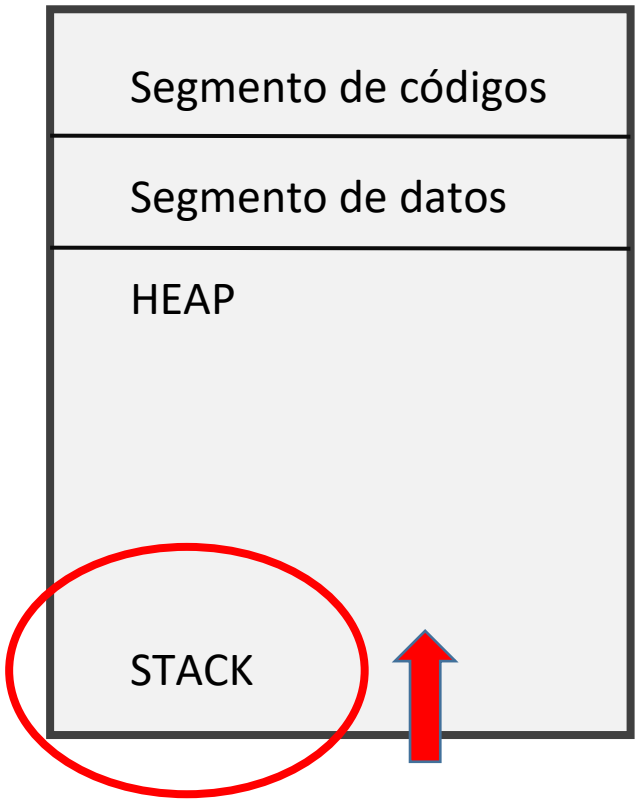
Las variables que se declaran **dentro** de una **función** tienen existencia en el **Stack.** Su alcance es **local**, cuando la función se extingue, la zona que ellas ocupan se elimina y todos los datos se pierden.

Comparación - Stack vs Heap

Stack	Heap
La asignación sucede en los llamados de funciones	La asignación puede suceder en cualquier momento
La Información es almacenada de forma secuencial	La Información es almacenada de aleatoria
El tamaño de las variables debe ser conocida al compilar	El tamaño de las variables pude ser desconocida
La memoria se libera automáticamente	Es tarea del programador liberar la memoria utilizada
De acceso rápido	De acceso más lento

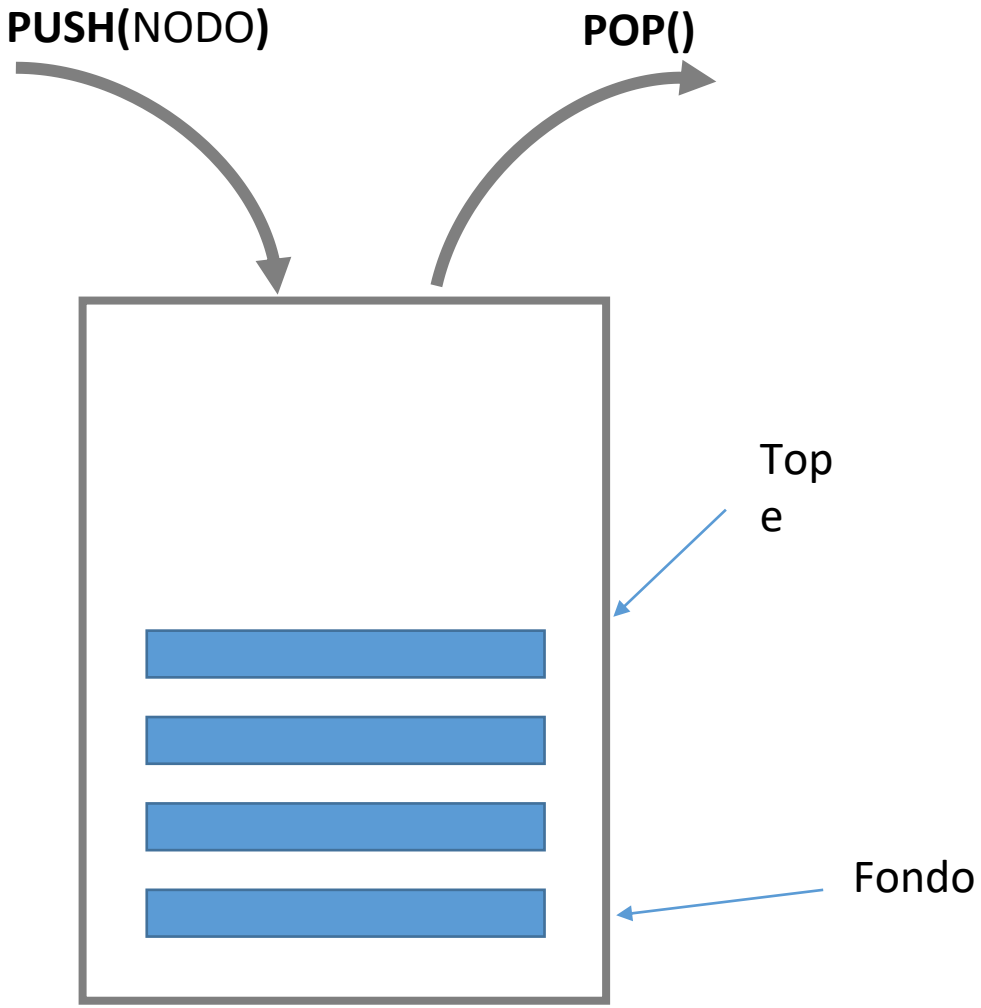
- Stack

Uso de memoria dinámica – stack



Tipo Pila

Estructura tipo LIFO (Last In First Out)

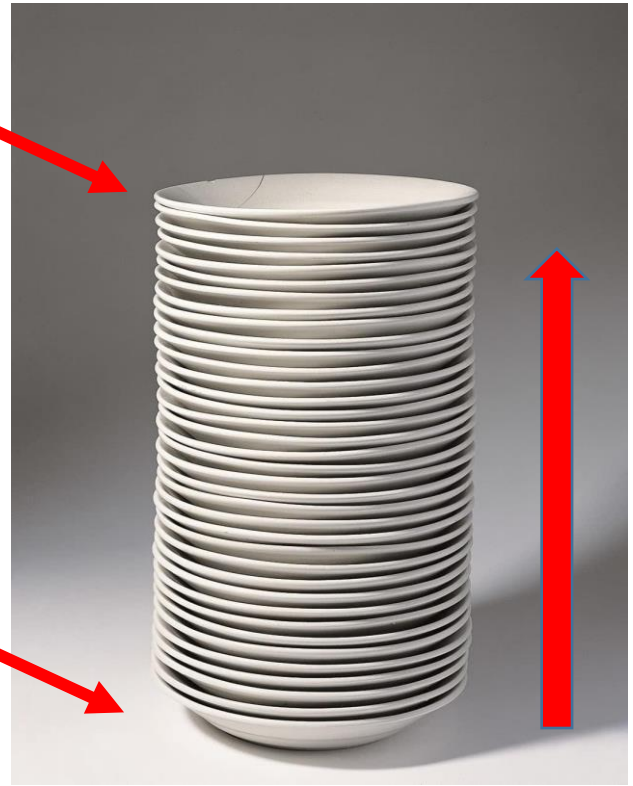


Uso de memoria dinámica – stack

Last In, First Out

Último en entrar

Primero en salir

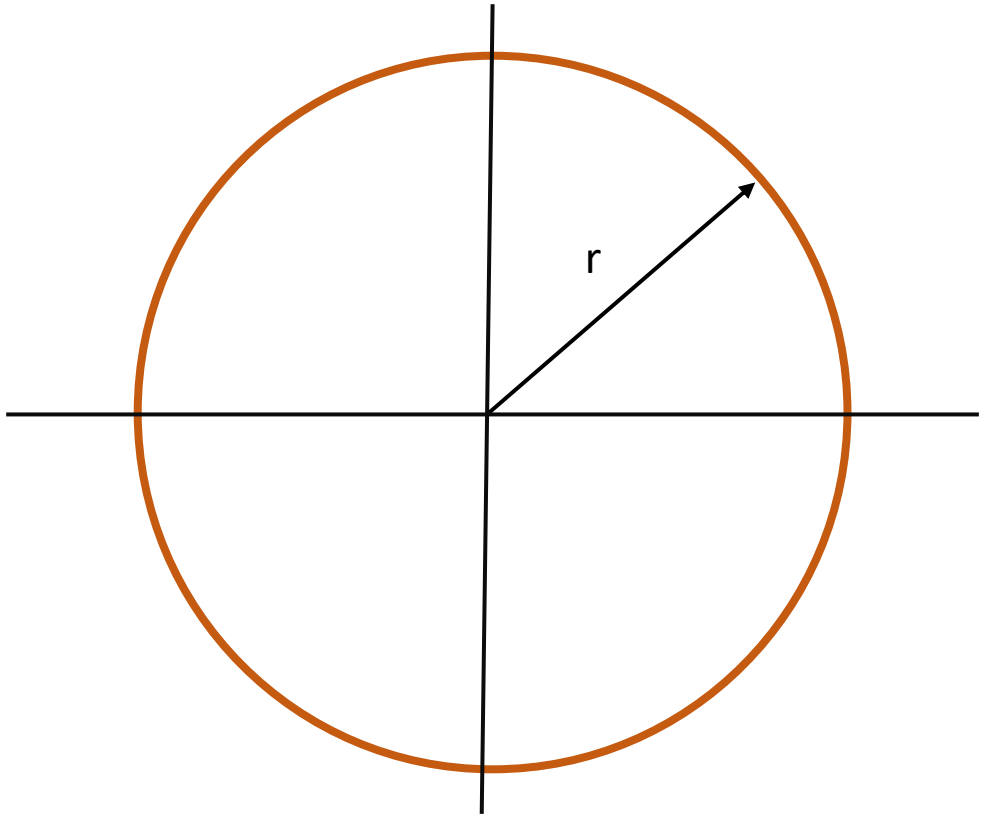


Primero en entrar

Último en salir

Uso de memoria dinámica – stack

Problema: escribir una aplicación para calcular el radio de una circunferencia centrada en el origen.

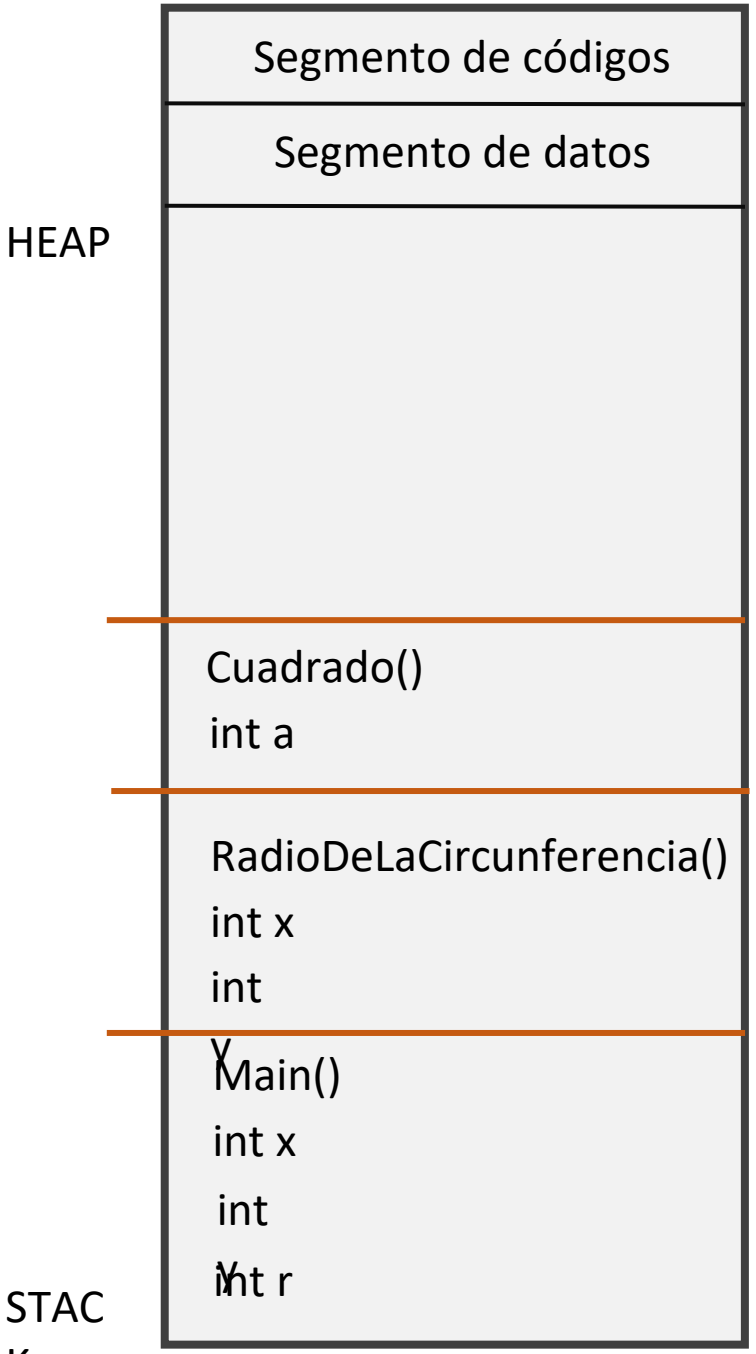


$$r^2 = x^2 + y^2$$

Uso de memoria dinámica – stack

<code/>

```c
#include <stdio.h>



int Cuadrado(int a)
{
    return a * a;
}


int RadioDeLaCircunferencia(int x, int y)
{
    return Cuadrado(x)  + Cuadrado(y);
}



int main()
{
    int x, y , r;
    r = RadioDeLaCircunferencia(x,y);
    printf("El radio de la circunferencia es: %d", r);
}
```



Ejemplo 00 - uso de recursión para el calculo del factorial de un número. (n!)

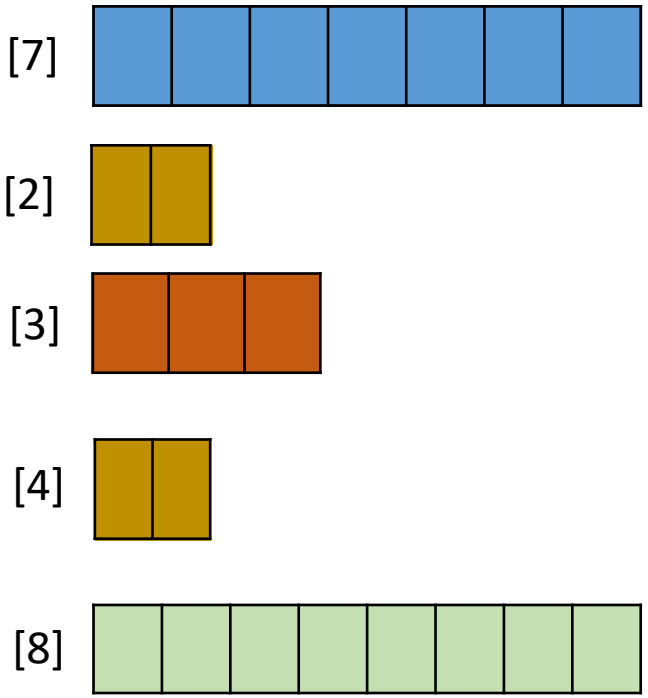
El factorial de un número n (denotado como n!) es el producto de todos los números positivos menores o iguales a n. Por definición, 0! = 1.

Algunos referencias interesantes – stack

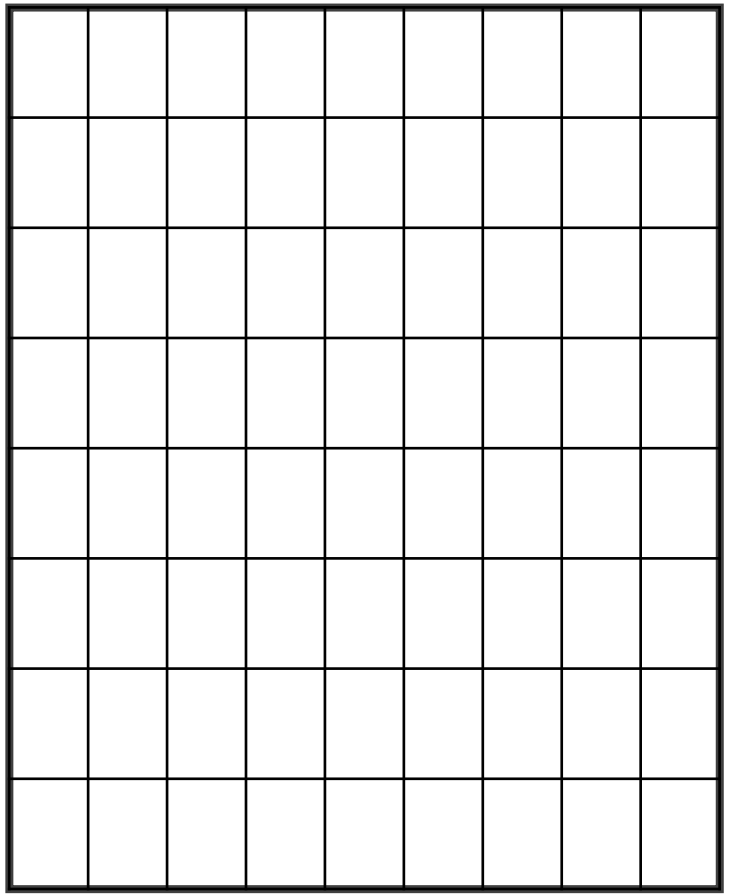
stackoverflow

- Heap

Uso de memoria dinámica – Heap



HEAP



Memoria dinámica – Asignación de memoria dinámica con c

- La biblioteca estándar de C proporciona funciones que **asignan y liberan memoria** de un bloque de memoria denominado el **montículo** (**HEAP**) del sistema.

En Lenguaje c (Están definidas en el archivo de cabecera stdlib.h)	
Las funciones para asignación de memoria dinámica	liberar memoria dinámica
• **malloc()** • **calloc()** • **realloc()**	• **Free()**

En lenguaje c++ (se utiliza un operador)	
Para asignación de memoria dinámica	liberar memoria dinámica
• Operador **new**	• Operador **delete**

Memoria Dinámica – Asignación de memoria dinámica con c

- **void** *malloc(size_t size);

tamaño del elemento

Solicita memoria y devuelve un puntero (void *)

```
int *array = malloc( 5 * sizeof(int));
```

- **void** *calloc(size_t nmemb, size_t size);

nº de elementos tamaño del elemento

Solicita memoria y devuelve un puntero (void *) e inicializa en 0 los elementos

```
int *array = calloc( 5, sizeof(int));
```

- **void** *realloc(void *ptr, size_t size);

Puntero inicial tamaño del elemento

Reasigna una porción de memoria reservada y devuelve un puntero (void *)

```
int *arr = malloc(2 * sizeof(int));
arr[0] = 1;
arr[1] = 2;
arr = realloc(arr, 3 * sizeof(int));
arr[2] = 3;
```

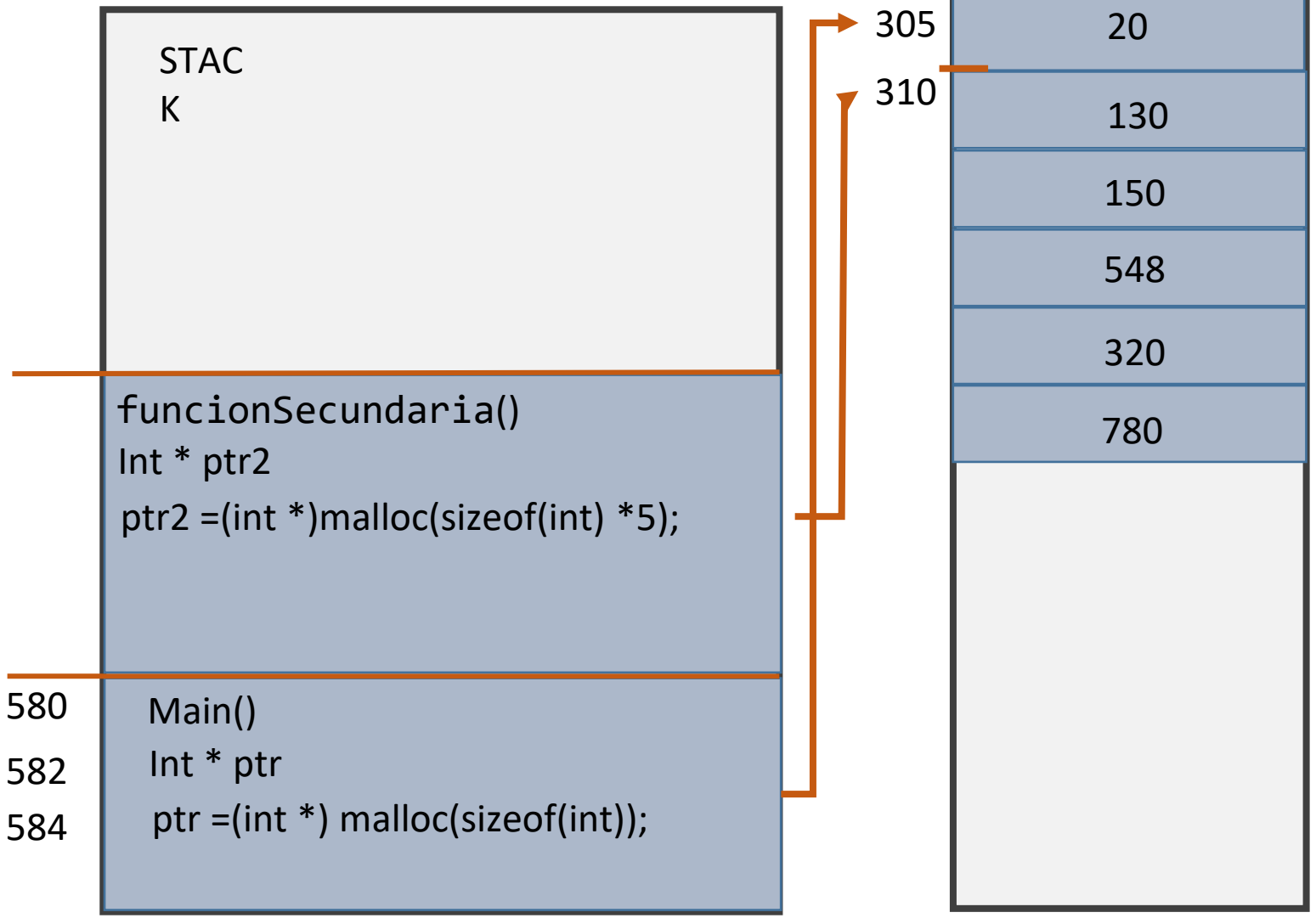
- **void free**(void *ptr);

Libera el bloque de memoria

Uso de memoria dinámica – heap

<code/>

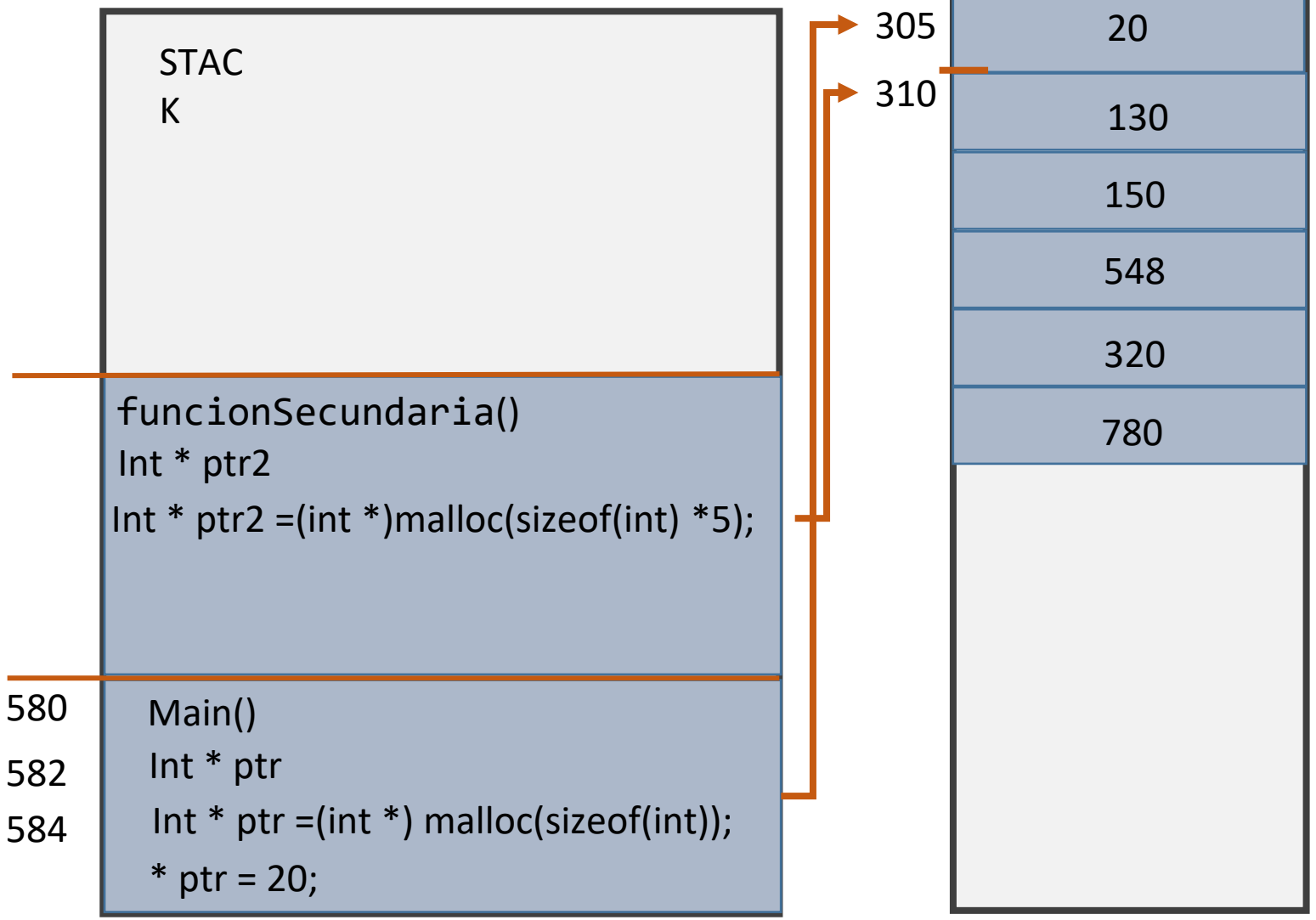
```c
#include <stdio.h>

void funcionSecundaria()
{
  int * ptr2;
  ptr2 = (int *) malloc(sizeof(int) * 5 );

  for(i=0;i<5;i++)
  {
    *(ptr2 +i) = 100+ rand()  % 801;
     printf("pVect[%d]: %d \r\n",i, ptr2[i]);
  }
}



int main()
{
    int * ptr;
    ptr = (int *) malloc(sizeof(int));
    *ptr = 20;
    printf("valor de ptr: %d", *ptr);
    funcionSecundaria();
    free(ptr);
}
```



Uso de memoria dinámica – heap

<code/>

```c
#include <stdio.h>

void funcionSecundaria()
{
  int * ptr2;
  ptr2 = (int *) malloc(sizeof(int) * 5 );

  for(i=0;i<5;i++)
  {
      ptr2[i] = 100+ rand()  % 801;
      printf("pVect[%d]: %d \r\n",i,*(ptr2+i));
  }
  free(ptr2);
}


int main()
{
    int * ptr;
    ptr = (int *) malloc(sizeof(int));
    *ptr = 20;
    printf("valor de ptr: %d", *ptr);
    funcionSecundaria();
    free(ptr);
}
```



Asignación dinámica de memoria
Una Visión desde el código

Memoria Dinámica – Asignación de memoria dinámica con c

Puntero simple

```
char * buff[50];
scanf("%s", buff); // usuario ingresa una cadena
int tamCadena = strlen(buff);

char * Cadena;
Cadena = (char *) malloc (tamCadena * sizeof(char) + 1);
```

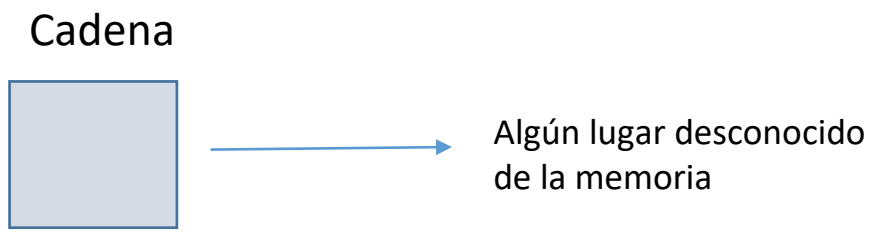
Puntero destino

Tipo del puntero

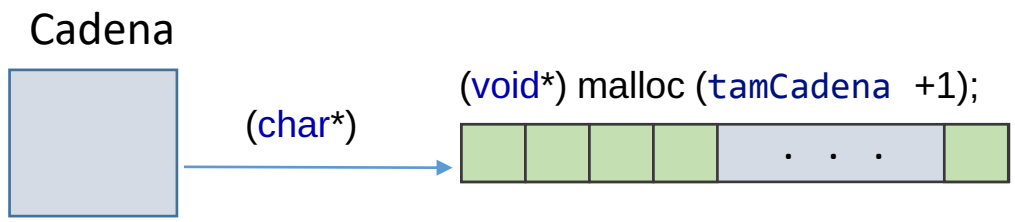
Cantidad de char que se requiere + 1

```
strcpy(Cadena, buff); // copio contenido de buff a cadena
printf("%s",Cadena); // imprimo cadena por pantalla
```

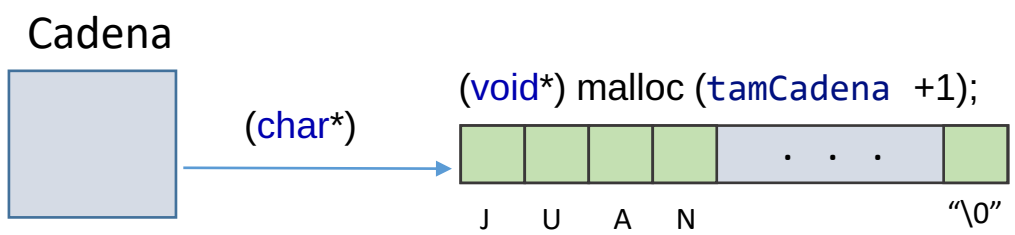
Situación Inicial



En el momento de la reserva



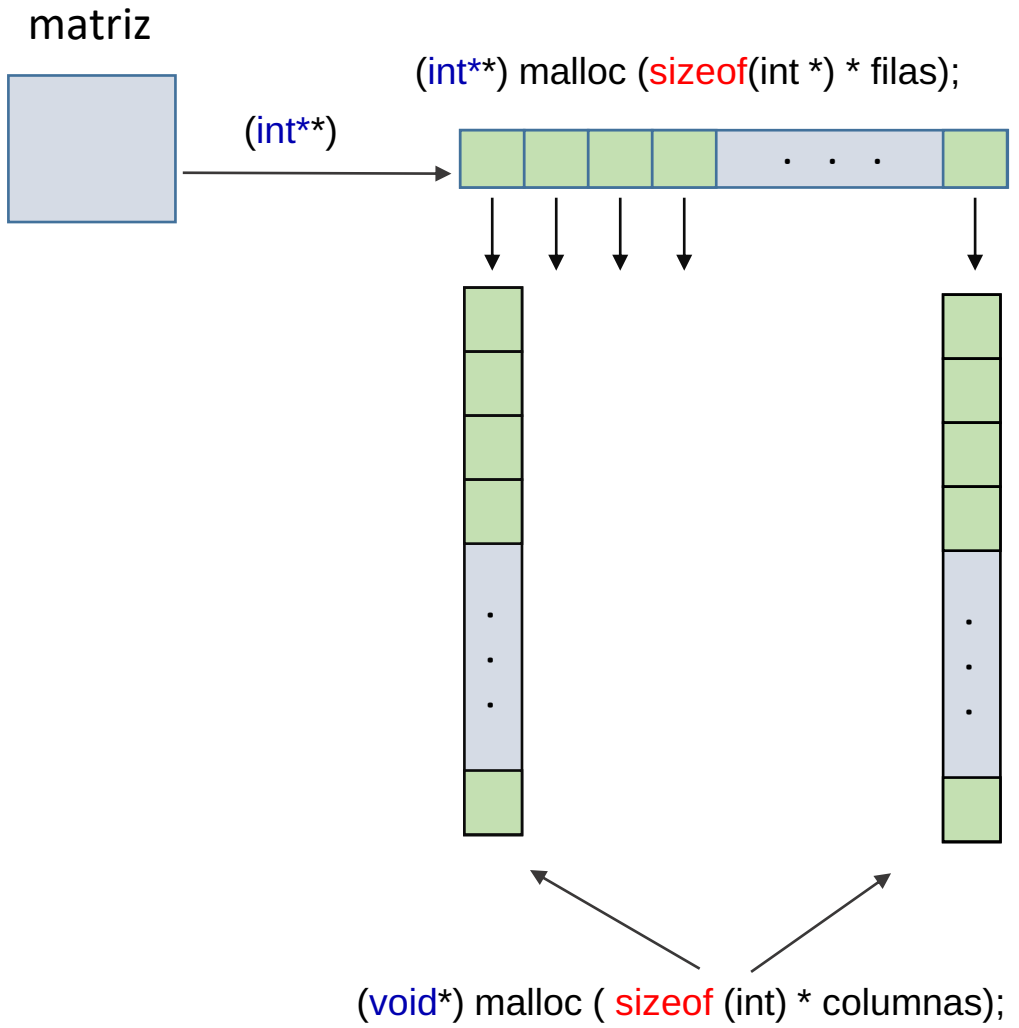
En el momento de la asignación strcpy(Cadena, buff)



Memoria Dinámica – Asignación de memoria dinámica con c

Puntero doble

```
int filas = 10, columnas = 10;
int ** matriz;

scanf ("%d", &filas);
scanf ("%d", &columnas);

matriz = (int **) malloc (sizeof(int *) * filas);

for (int i = 0; i < Columnas; i++)
{
        matriz[i] = (int *)malloc(columnas * sizeof(int));
        for (int j = 0; j < filas; j++)
        {
                matriz[i][j] = j;
        }
}
```



Memoria Dinámica – Asignación de memoria dinámica con c

Puntero de estructura y arreglo de estructura

```
typedef struct TRectang
{
    int Ancho;
    int Alto;
}TRectang;

int i;
TRectang * Figuras;

printf (“Tamaño del vector");
scanf ("%d", &cant);

Figuras  =    (TRectang *) malloc (sizeof (TRectang) * cant);
```