C Programming

Brief Introduction


t.me/cyberh99


cyberh99@protonmail.com


github.com/cyberh99

Índice

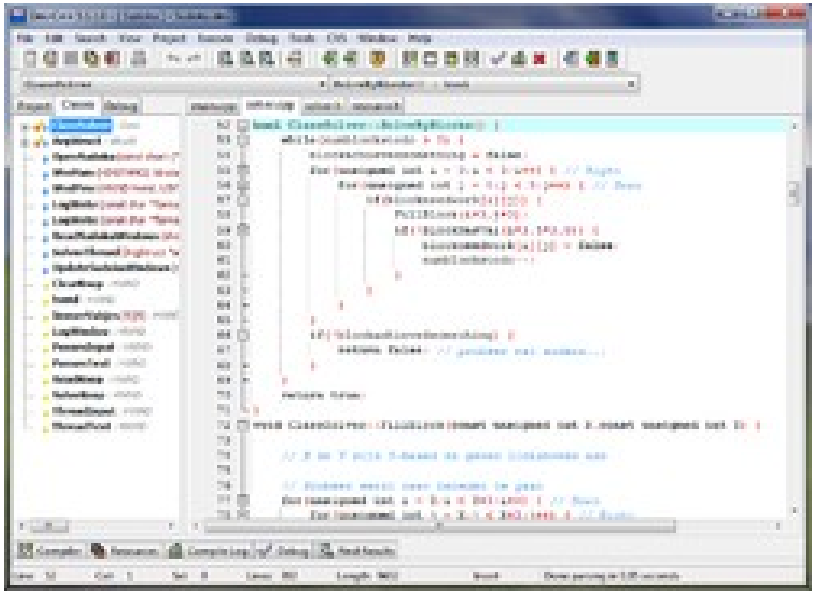
- Integrated Development Environment (IDE)
- Gcc para generar binarios
- I/O on C
- Funciones
- Punteros:
  - Definición de puntero
  - & || *
  - Erores en punteros
  - Punteros a Punteros
  - Vectores y String

Índice

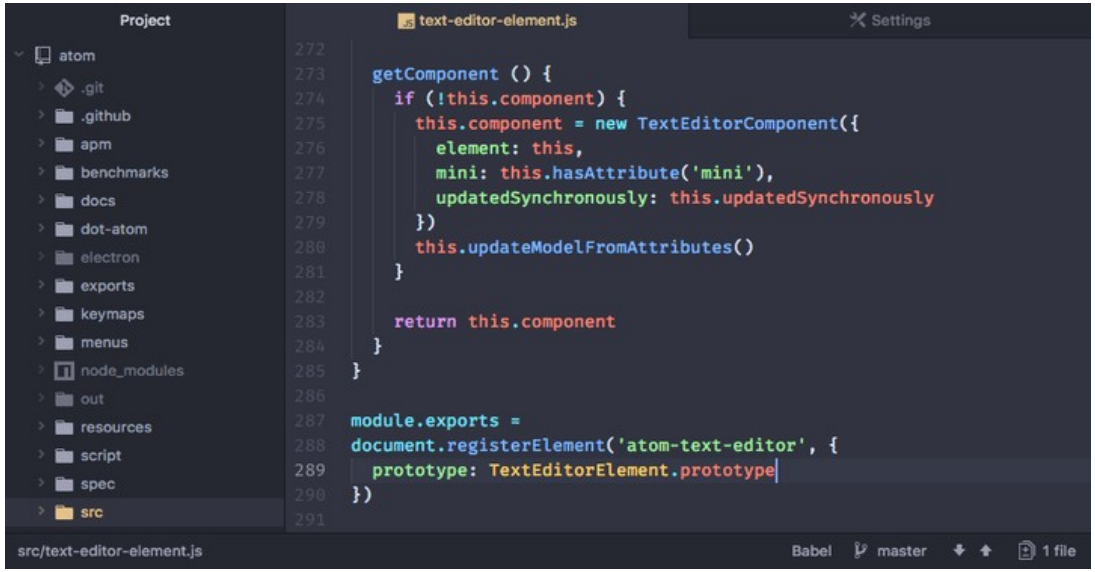
- Memoria dinámica:
  - Segmentación de Memoria
  - Funciones para trabajar la memoria
  - Separación de archivos
  - Creación de bibliotecas
- Structuras:
  - Define y typedef
  - Punteros a estructuras
  - Vectores de estructuras

Índice

Integrated Development Environment (I



Devc++



Atom

Gcc Usage

- Gcc -g program.c
  - Output: a.out
- Gcc -o programa//programa.exe program.c
  - Output: programa // programa.exe
- Gcc -g program.c
  - Output: a.out with debugging flags
- Gcc -c program.c
  - Output: Program.o for making libraries

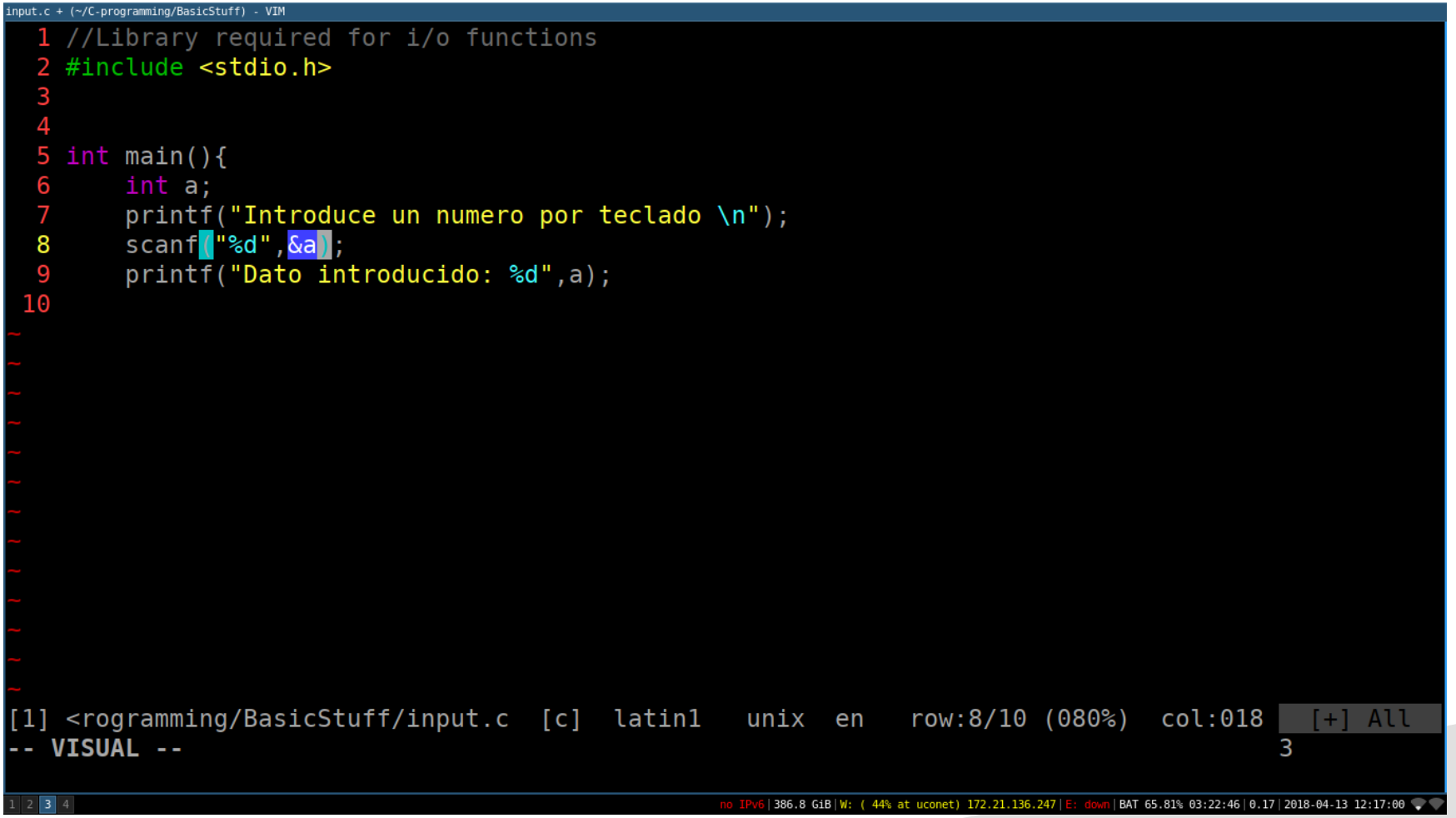
Already know...

- Tipos de variables
- If
- For, While, Do-While
- Funciones
- Arrays y Matrices (sin punteros)
- Argumentos
- I/O (Input/Output)

I/O Functions

- Data output:
  - Muestra stdout
  - Usa printf
- Data input:
  - Guarda la información en stdin
  - Usa scanf
  - Para string usar fgets

Printf / Scanf Example



Printf formmating

- %i → int
- %s → char
- %p → pointer
- %f → float
- %d → Double

- “\n”
- “\t”

Functions

- Función Main:
  - Siempre es entero
  - return 0;
- El tipo de la función depende del valor retornado:
  - Void
  - Int
  - Float
  - ...

¿Global o local?

- Variables locales:
  - Son accesibles por todas las funciones
- Variables globales:
  - Son accesibles exclusivamente por las funciones en las que son declaradas.

Functions example

functions.c (~/C-programming/BasicStuff) - VIM

```c
#include <stdio.h>


void PrintNumbers(){
    for (int i = 0;i<11;i++){
        printf("%d\t",i);
    }
}

int Module(int number1, int number2){
    int aux = number1%number2;
    return aux;
}

int main(){
    int number1 = 10, number2=9;
    PrintNumbers();
    printf("\n Module: %d \n",Module(number1,number2));
    return 0;
}
```

```
~
~
~
~
~
~
~
~
~
```

[1] ~/C-programming/BasicStuff/functions.c [c] latin1 unix en row:1/20 (005%) col:001 All

1 2 3 4 5 no IPv6 | 386.8 GiB | W: (44% at uconet) 172.21.136.247 | E: down | BAT 59.51% 03:07:22 | 0.10 | 2018-04-13 12:36:00

Pointers

Why use pointers?

- C was developed when computers were much less powerful than they are today and being very efficient with speed and memory usage was often not just desirable but vital. The raw ability to work with particular memory locations was obviously a useful option to have. A few tasks these days, such as programming microcontrollers, still need this. However most modern programmers do not need such fine control and the complications of using pointers make programs less clear to understand and add to the ways in which they can be go wrong. So why are pointers still used so much in C & its successor, C++?

Pointer

> “Una variable que contiene la dirección de memoria de otra”

* || & ?

- * → Hace referencia al contenido de una variable
- & → Hace referencia a la dirección de memoria de una variable

PointerExample1.c (~/C-programming/BasicStuff) - VIM

```
 1 #include <stdio.h>
 2
 3
 4
 5 int main(){
 6     int a = 19;
 7     int* b;
 8
 9     b = &a;
10     printf ("A Memory Address: %p \n",&a);
11     printf ("A Memory Address: %p\n ", b);
12     printf ("B Memory Address: %p\n",&b);
13     printf ("A Value:%d \n",*b);
14 }
~
~
~
~
~
~
~
~
~
[1] <ng/BasicStuff/PointerExample1.c  [c]  latin1   unix  en   row:12/14 (085%)  col:035   All
"PointerExample1.c" 14L, 238C
```

```
A Memory Address: 0x7fff71f59fec
A Memory Address: 0x7fff71f59fec
B Memory Address: 0x7fff71f59ff0
A Value:19
```

Practice 1 Exercise 6

6. Se desea mostrar la equivalencia entre funciones que devuelven un resultado y funciones que utilizan parámetros por referencia.

- a) Primera versión: función denominada minimo
  - 1. Recibe dos números num1 y num2 de tipo int pasados por valor.
  - 2. Devuelve como resultado el mínimo de los números num1 y num2 pasados como parámetros.
- b) Segunda versión: función denominada minimo_referencia
  - 1. Recibe dos números num1 y num2 de tipo int pasados por valor.
  - 2. Recibe otro parámetro denominado resultado de tipo int pero pasado por referencia.
  - 3. La función debe asignar a resultado el valor del mínimo de num1 y num2.
- c) Codifica un programa, denominado minimo.c, que permita comprobar el funcionamientolas dos funciones anteriores.

Paso por valor // Paso por referencia

- Paso por valor → Copiamos el contenido dentro de la función.
  - Consume más memoria
- Paso por referencia → Utilizamos punteros para pasar la dirección de memoria de los elementos
  - Es más útil cuando trabajamos con elementos complejos.

<code>

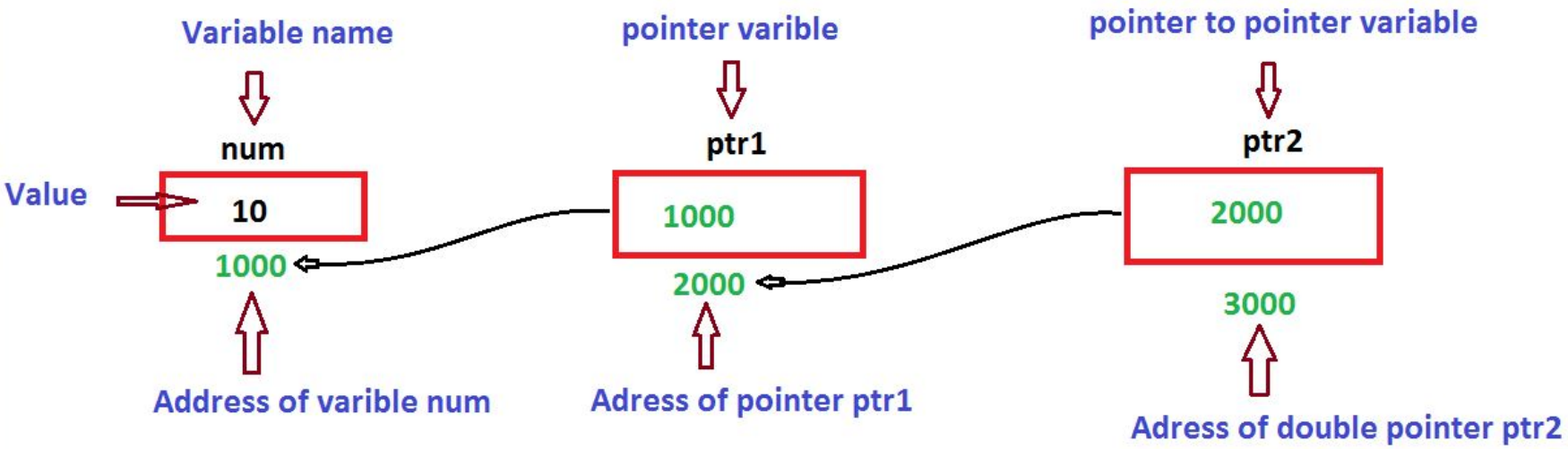
Common Errors

- Punteros no inicializados
- Punteros de diferente tipo
- Asignación de valores por referencia incorrectos

Pointer Types

- NULL
  - No apunta a nada, no tiene un valor válido en memoria.
  - Se traduce como 0.
- VOID
  - Puntero que apunta a cualquier tipo de dato
  - Surge cuando estamos utilizando funciones como malloc

Pointer to Pointer to Pointer ....



<code>

Vectores

<code>

VectorPointerExample.c + (~/C-programming/BasicStuff) - VIM

```
 1 #include <stdio.h>
 2 #include <time.h>
 3 #include <stdlib.h>
 4 #define  SIZE 10
 5
 6 void FillVector(int vector[]){
 7
 8     for (int i=0;i<SIZE;i++){
 9         srand(time(NULL));
10         *(vector+i)=rand()%2;
11          }
12 }
13
14 int main(){
15     int vector[SIZE];
16     FillVector(vector);
17     for (int i = 0; i<SIZE;i++){
18         printf("%i\n",vector+i);
19     }
20
21     return 0;
22 }
~
[1] <Stuff/VectorPointerExample.c  [c]  latin1   unix  en   row:16/22 (072%)  col:005   [+] All
1 more line; before #3  33 seconds ago
```

1 2 3 no IPv6 | 386.8 GiB | W: (100% at pipo) 192.168.43.144 | E: down | BAT 76.14% 04:22:19 | 0.04 | 2018-04-13 20:43:45

String

StringPointerExample.c (~/C-programming/BasicStuff) - VIM

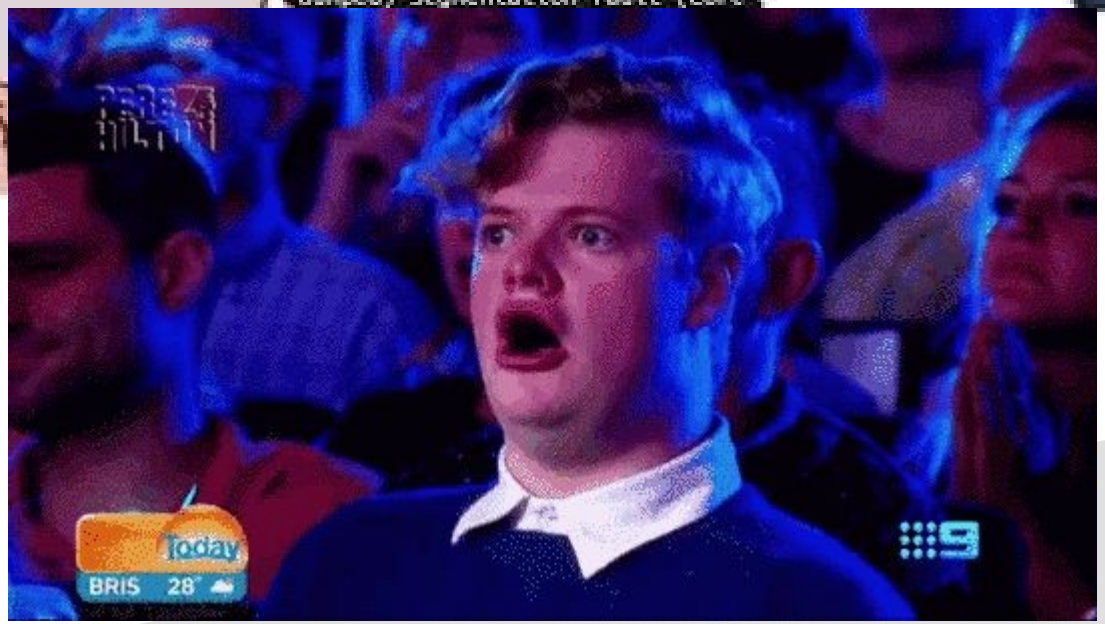
```c
#include <stdio.h>
#include <stdio.h>
#include <string.h>




int main(){
    char cadena[100];

    printf("introduce una cadena por teclado\n");
    fgets(cadena,100,stdin);
    printf("Cadena:\t%s",cadena);
    printf("%i\n",cadena[0]);
    for (int i =0 ; i<(strlen(cadena))+1;i++){
        printf("%i\n",cadena[i]);
    }
}
```

```
[1] ~/C-programming/BasicStuff/StringPointerExample.c  [c]  latin1   unix  en                    row:18/18 (100%)  col:001   All
```

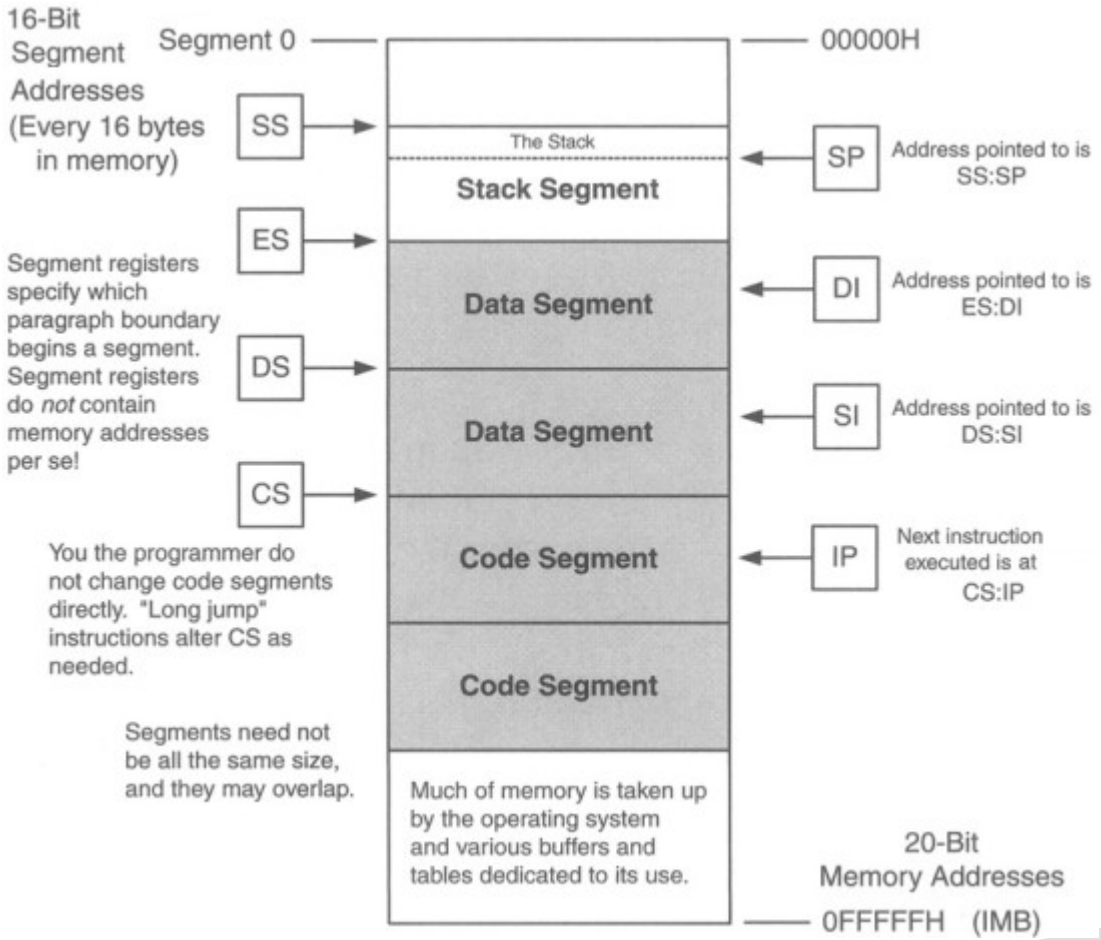
```
1 2 3 4 5 7   no IPv6 | 386.6 GiB | W: ( 82% at pipo) 192.168.43.144 | E: down | BAT 72.08% 03:01:10 | 0.06 | 2018-04-15 19:12:50
```

<code>

Memoria dinamica



Segmentación de memoria



<code>
</code>

Segmentos de Memoria

- Text Segment:
  - Es la parte encargada de almacenar el código compilado.
- Data Semgent:
  - Almacena Variables estáticas y globales
- BSS Segment:
  - Variables no inicializadas

<code>

¿Dónde se hace la reserva de memoria?

- Heap(montículo):
  - Parte libre de la memoria que puede ser modificada por el programador a su antojo.
- Stack(Pila):
  - Se encarga de almacenar variables de control y variables locales

Funciones para la reserva de memoria

- Reservar memoria:
  - Malloc
  - calloc
- Realojar memoria:
  - realloc
- Liberar memoria:
  - free

Reserva de memoria: malloc

- void *malloc(size_t size)
  - La reserva se hace en el heap
  - Esta parte de la memoria no puede ser reasignada
  - No inicializa la memoria
  - Devuelve un puntero void (hay que hacer casting)
  - Si no realiza correctamente las operaciones es NULL

<code>

Reserva de memoria: calloc

- void *calloc(size_t nelem, size_t size)
  - Reserva en el heap
  - Inicializa la memoria a cero
  - Se puede hacer casting del resultado
  - La parte de memoria asignada no puede ser usada en otra funcion
  - Si no realiza la memoria de forma exitosa devuelve NULL

<code>

Malloc vs Calloc

- Malloc simplemente reserva la memoria conservando los valores basura existentes en memoria, de forma que puede inducir a error cuando estamos trabajando con vectores o matrices.

- Calloc, por el contrario se encarga de inicializar la memoria a 0 de forma que es más cómodo detectar fallos, sin embargo calloc es más lento

Realojar memoria: realloc

- void *realloc(void *ptr, size_t size)
  - Devuelve la dirección del nuevo objeto.
  - Si el objeto antiguo es menor que el nuevo, copia todos los valores.
  - Si el objeto nuevo es menor rellena hasta donde pueda
  - Si la petición no tiene éxito devuelve NULL

<code>

Liberar memoria: Free

- void free(void *ptr)
  - Libera la memoria
  - No cambia el valor a NULL, ptr sigue apuntando a la memoria.
  - Si ptr es NULL no ha tenido efecto

Separación en archivos

- Main.c
  - Contiene exclusivamente llamadas a funciones de funciones.c
- Head.c
  - Contiene la declaración de:
    - Funciones
    - Typedef
    - Dedfine
    - #ifndef
    - Define
    - #endif
- Funciones.c
  - Funciones del programa

P2 → Ej2

Escribe una funcion que dado un vector dinamico, la longitud y un numero determinado, devuelva dos vectores dinamicos, de forma que uno contiene los elementos mayores y otro devuelve los elementos menores o iguales que el numero

Done?

Creación de bibliotecas

- Gcc -c funciones.c
- ar rsv LibreriaFinal funciones.o
- Gcc -o programa main.c LibreriaFinal
- Gcc -o programa main.c -L home…./LibreriaFinal

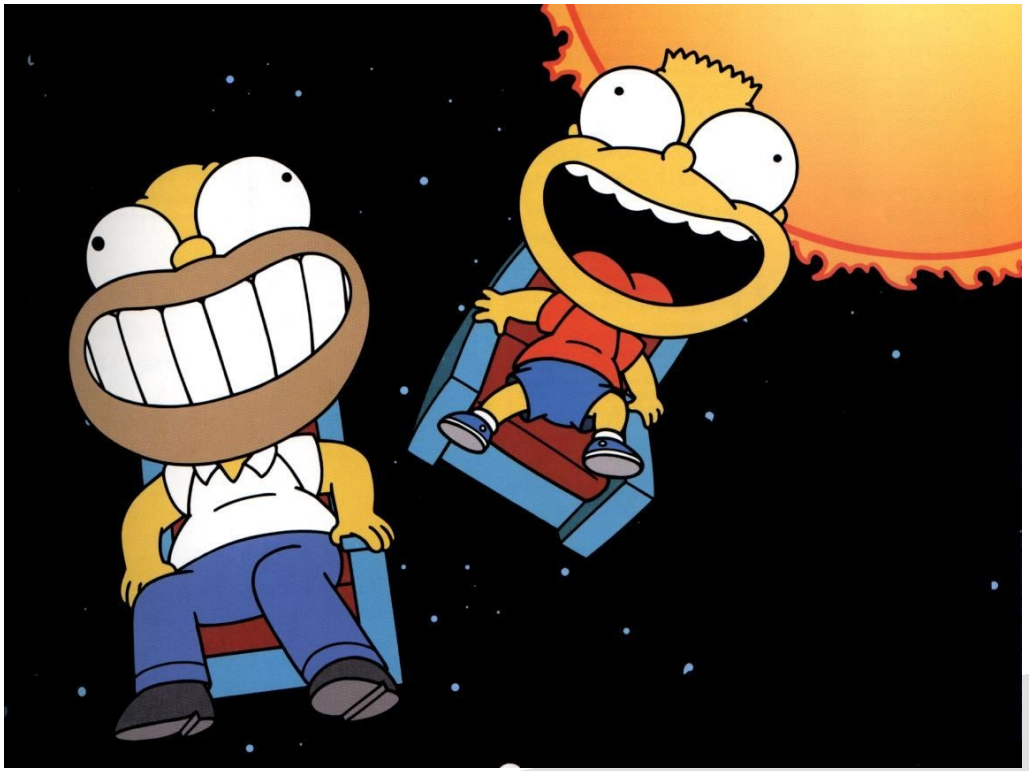
MinCol

4. Escribe un programa que implemente las siguientes funciones sobre matrices dinámicas y las llame de manera secuencial (no es necesario un menú) mostrando, de manera adecuada, la salida por pantalla.
   - *int ** reservarMemoria (int nFil, int nCol).*
     - Reserva memoria para una matriz de "nFil" filas y "nCol" columnas.
   - *void rellenaMatriz (int **matriz, int nFil, int nCol).*
     - Función que rellene una matriz con valores aleatorios en el intervalo [1,20].
   - *void imprimeMatriz (int **matriz, int nFil, int nCol).*
     - Función que imprima una matriz por pantalla.
     - Usa la notación de aritmética de punteros para recorrer la matriz.
   - *int * minCol (int **matriz, int nFil, int nCol).*
     - Función que devuelva un vector dinámico con los mínimos de cada columna.
   - *void liberarMemoria(int ***matriz, int nFil).*
     - Función para liberar la memoria de una matriz reservada dinámicamente.

Done?

Structuras

- structure is another user defined data type available in C that allows to combine data items of different kinds



Struct

- Struct Alumno{
  - Int numero;
  - Char nombre[100];
  - Char apellido[100];
  - Float peso;
- };

Typedef Struct

- Typedef Struct Alumno{
  - Int numero;
  - Char nombre[100];
  - Char apellido[100];
  - Float peso;
- }DatosAlumno;

Define

- Nos permite asignar un valor determinado a una palabra:
  - Define SIZE 100
  - Suelen incluirse en la cabecera

Punteros a estructuras → Operador "→

- Es usado cuando estamos trabajando con punteros a estructuras.
- Es necesario hacer la reserva de memoria

<code>

Vectores de estructuras
p2 → ej3

Escribe un programa que permita gestionar los jugadores de baloncesto del equipo de una ciudad.

Para ello se guardará la información de cada jugador en la siguiente estructura:

```
struct Ficha_jugador {

char nombre[50];

int dorsal;

 float peso;

int estatura;};
```

Done?

That's All

"Programar es como el sexo,
un error y das soporte por
el resto de tu vida"