Tutorial de C;)

1. Elementos obsoletos, pero aun validos:

Ha evolucionado, algunos elementos se consideran obsoletos pero siguen siendo válidos, como:

* gets(): Se utilizaba para leer cadenas, pero es inseguro debido a la falta de control sobre la longitud de la entrada y se recomienda usar fgets() en su lugar.
* register: Aunque es una sugerencia para almacenar variables en registros, el compilador moderno decide automáticamente la mejor ubicación.

Ejemplo:

#include <stdio.h>

int main() {

char buffer[100];

// Se considera inseguro

gets(buffer); // No usar

printf("Entrada: %s\n", buffer);

return 0;

}

1. Nuevas Palabras Claves:

Se introdujeron nuevas palabras claves, como:

* inline: Sugerencia al compilador para que inserte el código de la función directamente en el lugar donde se llama.
* restrict: Indica que un puntero es el único acceso a un objeto, permitiendo optimizaciones.

Ejemplo:

#include <stdio.h>

inline int suma(int a, int b) {

return a + b;

}

int main() {

printf("Suma: %d\n", suma(3, 4));

return 0;

}

1. Convenciones de Llamada:

Determinan cómo se pasan los argumentos a las funciones y cómo se devuelve el control, como:

* CDECL: Usada en sistemas x86, permite que las funciones reciban un número variable de argumentos.
* STDCALL: Utilizada para llamadas a funciones de API, el llamador limpia la pila.

Ejemplo:

#include <stdio.h>

void \_\_cdecl miFuncion(int a) {

printf("Valor: %d\n", a);

}

int main() {

miFuncion(10);

return 0;

}

1. Pilas y Recursividad:

* Pilas: Estructuras de datos que siguen el principio LIFO (Last In, First Out). En C, se pueden implementar utilizando arrays o listas enlazadas.
* Recursividad: Una función se llama a sí misma. Es crucial tener una condición de salida para evitar bucles infinitos.

Ejemplo:

#include <stdio.h>

int factorial(int n) {

if (n == 0) return 1;

return n \* factorial(n - 1);

}

int main() {

printf("Factorial de 5: %d\n", factorial(5));

return 0;

}

1. Memoria y Cadenas:

* Memoria: Realiza manualmente usando malloc(), calloc(), realloc(), y free().
* Cadenas: Se representan como arrays de caracteres terminados en '\0'. Funciones como strcpy(), strlen(), y strcat() son esenciales para manipular cadenas.

Ejemplo:

#include <stdio.h>

#include <stdlib.h>

int main() {

char \*cadena = (char \*)malloc(100 \* sizeof(char));

if (cadena != NULL) {

strcpy(cadena, "Hola, mundo!");

printf("%s\n", cadena);

free(cadena);

}

return 0;

}

1. Procesos, Subprocesos e Hilos:

- Procesos: Un proceso es una instancia de un programa en ejecución. Se gestionan mediante llamadas al sistema como fork().

* Subprocesos (Threads): Permiten la ejecución concurrente dentro de un proceso. Se pueden crear usando la biblioteca pthread.
* Hilos: Permiten que un programa realice múltiples tareas simultáneamente, mejorando la eficiencia.

Ejemplo:

#include <stdio.h>

#include <pthread.h>

void\* miHilo(void\* arg) {

printf("Hola desde el hilo!\n");

return NULL;

}

int main() {

pthread\_t hilo;

pthread\_create(&hilo, NULL, miHilo, NULL);

pthread\_join(hilo, NULL);

return 0;

}

1. Network Sockets:

Permiten la comunicación entre procesos en diferentes máquinas a través de una red, se utilizan funciones como socket(), bind(), listen(), accept(), connect(), y send()/recv() para gestionar la comunicación.

Ejemplo:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

int main() {

int socket\_desc;

struct sockaddr\_in server;

socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

if (socket\_desc == -1) {

printf("No se pudo crear el socket\n");

}

server.sin\_family = AF\_INET;

server.sin\_port = htons(8888);

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

if (connect(socket\_desc, (struct sockaddr \*)&server, sizeof(server)) < 0) {

printf("Conexión fallida\n");

return 1;

}

printf("Conectado\n");

close(socket\_desc);

return 0;

}

Infografía sobre Git y GitHub

**¿Qué es Git?**

Es un sistema de control de versiones distribuido que permite rastrear cambios en archivos y coordinar el trabajo entre múltiples personas.

## ¿Qué es GitHub?

GitHub es una plataforma de desarrollo colaborativo que utiliza Git para el control de versiones y permite alojar proyectos en la nube.

Comandos para Git

* git init - Inicializa un nuevo repositorio Git.
* git clone [url] - Clona un repositorio remoto.
* git add [archivo] - Agrega archivos al área de preparación.
* git commit - Guarda los cambios en el repositorio.
* git push - Envía los cambios al repositorio remoto.
* git pull - Descarga y fusiona cambios del repositorio remoto.

Comandos para GitHub

* git fork - Crea una copia de un repositorio en tu cuenta de GitHub.
* git pull request - Solicita que se revisen y fusionen tus cambios en el repositorio original.
* git merge - Combina cambios de diferentes ramas.