Tutorial sobre el Lenguaje de Programación C

1. Elementos Obsoletos, pero Aún Válidos

El lenguaje C ha evolucionado desde su creación, pero algunos elementos antiguos siguen siendo válidos y utilizados por razones de compatibilidad y simplicidad.

Ejemplos:

Funciones gets() y strcpy(): Aunque son consideradas inseguras debido a problemas de desbordamiento de buffer, todavía se utilizan en algunos contextos.

Declaraciones de variables en el inicio de bloques: Antes de C99, las variables debían declararse al inicio de un bloque de código. Aunque ahora se pueden declarar en cualquier lugar, esta práctica aún se observa en código antiguo.

#include <stdio.h>

int main() {

char name[50];

gets(name); // Obsoleto y no seguro

printf("Hello, %s\n", name);

}

2. Nuevas Palabras Claves

Desde el estándar C99 y posteriores, se han introducido nuevas palabras clave.

Ejemplos:

inline: Sugerencia al compilador para insertar el cuerpo de una función directamente en el código llamador.

\_Bool: Introduce el tipo de dato booleano.

restrict: Indica que el puntero es el único que accede al objeto al que apunta durante su tiempo de vida.

#include <stdio.h>

inline int add(int a, int b) {

return a + b;

}

int main() {

int sum = add(3, 4);

printf("Sum: %d\n", sum);

}

3. Convenciones de Llamada

Las convenciones de llamada determinan cómo los parámetros se pasan a las funciones y cómo se gestionan las pilas.

cdecl: La convención por defecto en la mayoría de los compiladores para C.

stdcall: Utilizado principalmente en Windows API.

#include <stdio.h>

void \_\_cdecl myFunction() {

printf("Called with cdecl convention\n");

}

void \_\_stdcall myStdFunction() {

printf("Called with stdcall convention\n");

}

int main() {

myFunction();

myStdFunction();

}

4. Pilas y Recursividad

La recursividad es una técnica donde una función se llama a sí misma. Las pilas juegan un papel crucial en la gestión de estas llamadas.

Ejemplo:

Factorial de un número utilizando recursividad.

#include <stdio.h>

int factorial(int n) {

if (n == 0) return 1;

return n \* factorial(n - 1);

}

int main() {

int num = 5;

printf("Factorial of %d is %d\n", num, factorial(num));

}

5. Memoria y Cadenas

La gestión de memoria y el manejo de cadenas son fundamentales en C.

Ejemplo:

Asignación dinámica de memoria y manipulación de cadenas.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main() {

char \*str;

str = (char \*) malloc(15);

strcpy(str, "Hello, World!");

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

free(str);

}

6. Procesos, Subprocesos e Hilos

C permite la creación de procesos y subprocesos (threads) para ejecución concurrente.

Ejemplo:

Creación de hilos utilizando la librería pthread.

#include <stdio.h>

#include <pthread.h>

void \*printMessage(void \*message) {

printf("%s\n", (char \*)message);

return NULL;

}

int main() {

pthread\_t thread;

char \*message = "Hello, Thread!";

pthread\_create(&thread, NULL, printMessage, (void \*)message);

pthread\_join(thread, NULL);

}

7. Network Sockets

C es ampliamente utilizado en programación de red a través de sockets.

Ejemplo:

Cliente simple que se conecta a un servidor.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

int main() {

int sock;

struct sockaddr\_in server;

char message[1000], server\_reply[2000];

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

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

server.sin\_family = AF\_INET;

server.sin\_port = htons(8888);

connect(sock, (struct sockaddr \*)&server, sizeof(server));

printf("Connected\n");

strcpy(message, "Hello, Server");

send(sock, message, strlen(message), 0);

recv(sock, server\_reply, 2000, 0);

printf("Server reply: %s\n", server\_reply);

close(sock);

}