LENGUAJE C

PROF. YULITH VANESSA ALTAMIRANO FLORES

TALLER 5
CLASES DE ALMACENAMIENTO

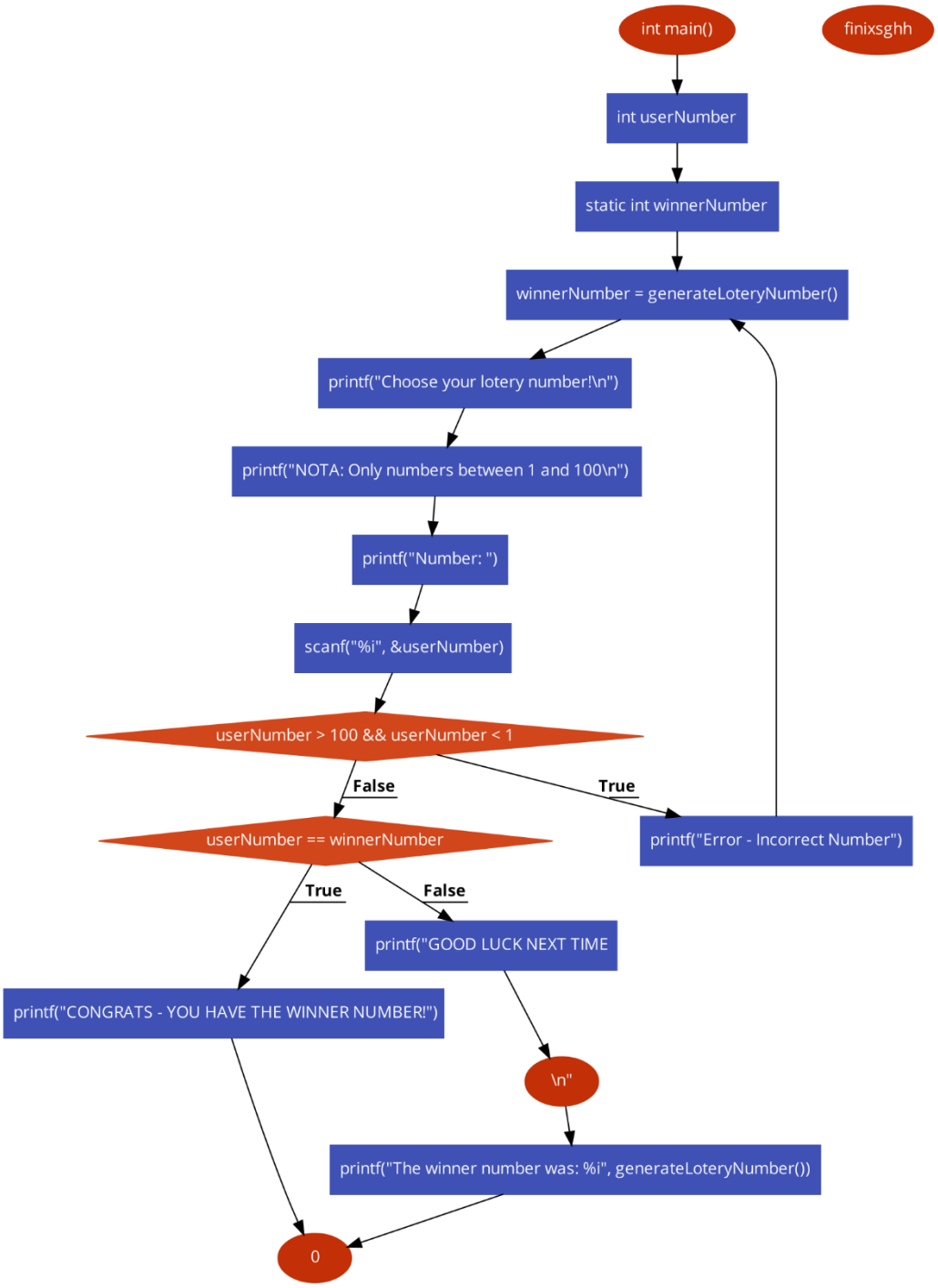
KEVIN ALEJANDRO GONZALEZ TORRES
GRUPO 932

REPOSITORIO

https://github.com/keevin-21/KAGT_Lenguaje_C_932

DIAGRAMA DE FLUJO

Actividad 1:



1. Declara una variable automática llamada contador en una función. Incrementa su valor en un bucle y muestra su valor en cada iteración.

```c
#include<stdio.h>

int main()
{
    for (int contador = 0; contador < 10; contador++)
    {
        printf("%i\n", contador);
    }
    return 0;
}
```

```
0
1
2
3
4
5
6
7
8
9
PS G:\My Drive\UABC\TercerSemestre\KAGT_Lenguaje_C_932\Talleres\Taller_5>
```

¿Qué sucede con la variable al salir de la función?

Al salir de la función main(), la variable contador ya no existe y no afecta a otros aspectos del programa.

2. Declara una variable externa llamada saldo en un archivo fuente (archivo.c) y accede a ella desde otro archivo fuente (otroarchivo.c). Modifica su valor en ambos archivos y muestra el valor final. ¿Cómo afecta la visibilidad y el tiempo de vida de la variable externa?

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

void saldos(int cantidad);

int main()
{
    extern int saldo;
    int cantidad;

    printf("SALDO ACTUAL: %d\n", saldo);
    printf("INGRESA LO QUE QUIERAS AGREGAR A SALDO: ");
    scanf("%d", &cantidad);
    saldos(cantidad);
}
```

```c
//saldo.cpp

#include <stdio.h>
#include <stdlib.h>

int saldo = 1000;

void saldos(int cantidad)
{
    saldo+= cantidad;
    printf("VALOR DE SALDO ACTUAL: %d\n ", saldo);
    system("PAUSE");
}
```

```
SALDO ACTUAL: 1000
INGRESA LO QUE QUIERAS AGREGAR A SALDO: 440
VALOR DE SALDO ACTUAL: 1440
 Press any key to continue . . .
```

3. Declara una variable estática llamada contador en una función y muestra su valor en cada llamada a la función.

```c
#include <stdio.h>

static int contador = 0;

void incrementarYMostrarContador() {
    contador++;
    printf("El valor del contador es: %d\n", contador);
}

int main() {

    incrementarYMostrarContador();
    incrementarYMostrarContador();
    incrementarYMostrarContador();
    incrementarYMostrarContador();
    incrementarYMostrarContador();
    incrementarYMostrarContador();
    incrementarYMostrarContador();
    incrementarYMostrarContador();
    incrementarYMostrarContador();
    incrementarYMostrarContador();

    return 0;
}

```

¿Qué sucede con la variable al salir de la función?

Dado que la variable contador es estática, su tiempo de vida se extiende más allá de la función incrementarYMostrarContador(). En otras palabras, la variable contador conservará su valor entre llamadas a la función y continuará existiendo en todo el programa incluso después de que main() haya terminado su ejecución.

¿Cómo difiere de una variable automática?

La diferencia principal entre una variable estática y una variable automática (también conocida como variable local) radica en su tiempo de vida y su ámbito

4. Declara una variable de registro llamada temp y otra automática llamada valor en una función. Compara el acceso y el tiempo de vida de estas variables.

```c
#include <stdio.h>

int main() {
    register int temp = 10;
    int valor = 20;

    printf("Valor de 'temp' (registro): %d\n", temp);
    printf("Valor de 'valor' (automática): %d\n", valor);

    {
        int valor = 30; // Otra variable 'valor' en un ámbito interno
        printf("Valor de 'valor' en ámbito interno: %d\n", valor);
    }

    // 'temp' sigue siendo accesible aquí
    printf("Valor de 'temp' después del ámbito interno: %d\n", temp);

    return 0;
}
```

¿Por qué usarías una variable de registro en lugar de una variable automática?

Evitar acceso a la memoria principal: Los registros de la CPU son más rápidos de acceder que la memoria principal (RAM). En algunas aplicaciones que requieren una ejecución extremadamente rápida, el acceso a registros puede ser beneficioso.

5. Declara una variable global llamada pi con un valor de 3.14159 y otra variable local con el mismo nombre en una función (Con diferente valor). Intenta acceder a ambas variables desde diferentes partes del programa.

```c
#include <stdio.h>

// Declaración de una variable global llamada 'pi'
float pi = 3.14159;

// Función que declara una variable local 'pi' con un valor diferente
void cambiarValorPi() {
    float pi = 3.14159265359; // Variable local con un valor diferente
    printf("Valor local de 'pi' dentro de la función: %.10f\n", pi);
}

int main() {
    printf("Valor global de 'pi': %.5lf\n", pi);

    // Llama a la función para cambiar el valor local de 'pi'
    cambiarValorPi();

    // Accede nuevamente a la variable global 'pi' desde 'main'
    printf("Valor global de 'pi' después de llamar a la función: %.5lf\n", pi);

    return 0;
}

```

¿Cuál es el resultado? Explica el concepto de ámbito y visibilidad.

- Valor global de 'pi': 3.14159
- Valor local de 'pi' dentro de la función: 3.1415927410
- Valor global de 'pi' después de llamar a la función: 3.14159

  El ámbito de una variable se refiere a la región del código donde esa variable es válida y accesible.

  La visibilidad se refiere a la capacidad de acceder a una variable desde un lugar particular en el código.