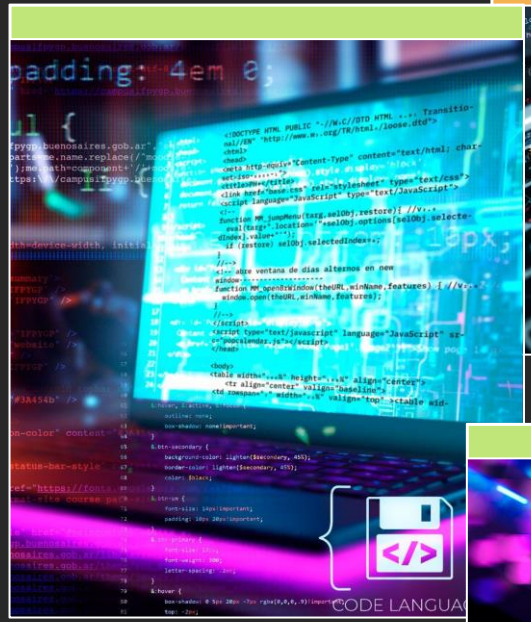
Programacion II [PRACTICA]

1 0 0 1 0 1 0 1 0 1 0 1 0 0

1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0



“HOLA MUNDO”

1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1 0

1 0 1 0 1 0 1 1 1 0 0 0 1 1 1 0

Imprimir una línea de Texto `[std::cout] •••

```cpp
// Programa para imprimir texto.
#include <iostream> // permite al programa imprimir datos en la pantalla
// la función main comienza la ejecución del programa
int main()
 {
std::cout << "Bienvenido a C++!\n"; // muestra un mensaje

return  0; // indica que el programa terminó con éxito
}
```

Pertenece a: La biblioteca estándar de C++ (iostream).

Utiliza operadores de flujo (<<) para enviar datos a la salida estándar.

std::cout es más seguro en términos de tipos de datos, ya que no requiere especificar un formato.

Se adapta automáticamente al tipo de dato que se imprime.

1 1 0 0 1 0 1 0 1 0 1 0 1 0 1 1 1 0

Secuencias de Escape

Secuencia de Escape	Descripción
\n	Nueva línea. Coloca el cursor al inicio de la siguiente línea.
\t	Tabulador horizontal. Desplaza el cursor hasta la siguiente posición de tabulación
\r	Retorno de carro. Coloca el cursor de la pantalla al inicio de la línea actual; no avanza a la siguiente línea.
\a	Alerta. Suena la campana del sistema.
\\	Barra diagonal inversa. Se usa para imprimir un carácter de barra diagonal inversa.
\'	Comilla sencilla. Se usa para imprimir un carácter de comilla sencilla
\"	Doble comilla. Se usa para imprimir un carácter de doble comilla.

Ingreso de Datos [std::cin]

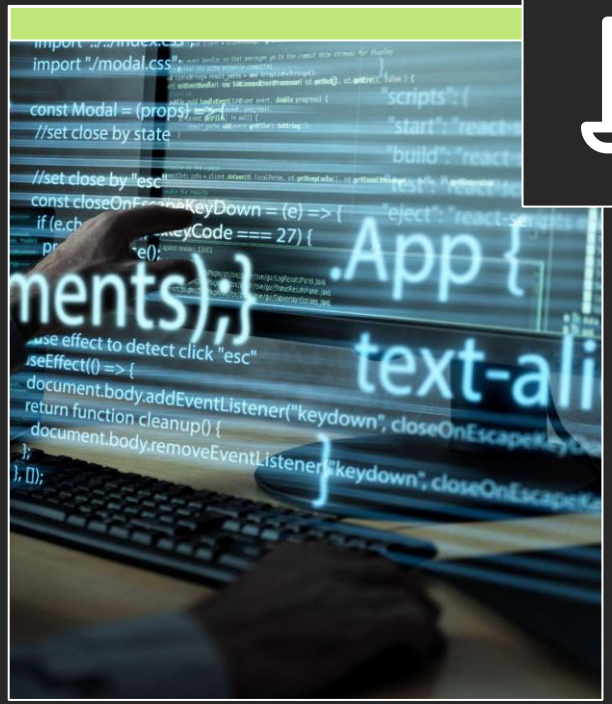
```cpp
#include <iostream>
// la función main comienza la ejecución del programa
int main()
{
// Declaraciones de variables
int numero = 0; // primer entero a sumar (se inicializa con 0)
std::cout << "Escriba un numero "; // pide los datos al usuario
std::cin >> numero;         // lee el primer entero del usuario y lo coloca en numero1

std::cout << "El numero ingresado es: " << numero << std::endl; // muestra la suma; fin de línea

return  0; // indica que el programa terminó con éxito
}
```

Pertenece a: La biblioteca estándar de C++ (iostream).

Estilo: Usa operadores de extracción (>>) para leer datos desde la entrada estándar (como el teclado) y almacenarlos en variables.

Maneja automáticamente los tipos de datos. No necesitas especificar el tipo de datos que estás leyendo, ya que se deduce del tipo de la variable.

1 1 0 0 1 0 1 0 1 0 1 0 1 0 1 1 1 0

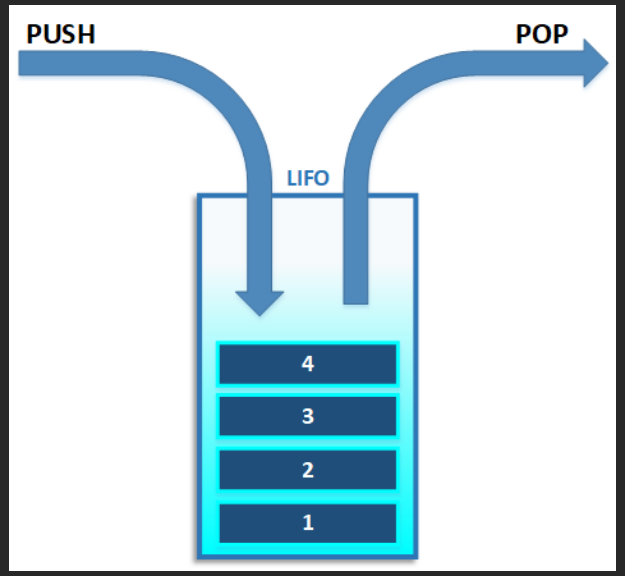


Memoria Dinámica

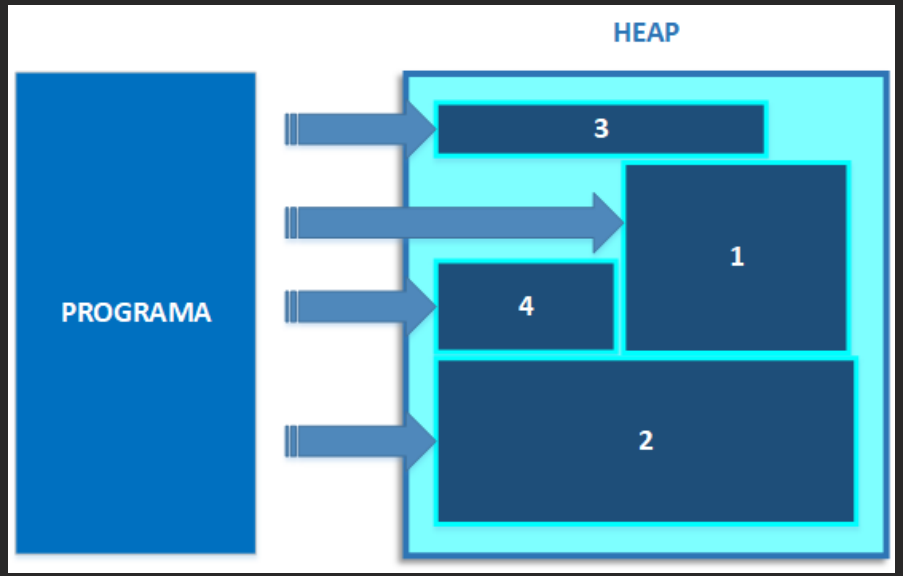
1 0 1 0 1 0 1 1 1 0 0 0 1 1 1 0

1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 0

Memoria Dinámica



PILA [Stack]



Montón [Heap]

Memoria Dinámica PILA [Stack]

- Las variables en la pila son locales a la función en la que se declaran.
- Su duración está limitada al tiempo de ejecución de la función en la que fueron creadas. Cuando la función termina, las variables en la pila se destruyen automáticamente.
- El ciclo de vida de la variable está atado al ciclo de vida de la función. No puedes extender la vida útil de una variable en la pila más allá de la función.
- La pila es administrada automáticamente. El sistema operativo asigna y libera memoria automáticamente cuando las funciones entran y salen.
- Esto es más rápido y seguro, pero limitado en tamaño. La pila tiene un tamaño fijo, y puedes provocar un "stack overflow" si asignas demasiada memoria.

1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0

Memoria Dinámica Montón [Heap]

- Aunque la variable que apunta a la memoria en el montón puede ser local a una función, la memoria en sí en el montón tiene un alcance global en cuanto a su duración; puede existir más allá de la función que la creó, si se mantiene una referencia válida.
- La memoria asignada en el montón persiste hasta que es liberada manualmente con delete, o hasta que el programa termina. No se libera automáticamente al salir de la función.
- Tienes control total sobre cuándo la memoria en el montón se libera. Puedes hacer que la memoria asignada en el montón dure tanto como quieras, siempre que tengas un puntero válido a esa memoria.

1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 0

Memoria Dinámica Montón [Heap]

- El montón es administrado manualmente. Tienes que usar new para asignar memoria y delete para liberarla.

- El montón tiene un tamaño mucho más grande y flexible, lo que permite asignar grandes bloques de memoria en tiempo de ejecución, pero es más lento y propenso a errores como fugas de memoria.

1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1 1 0

Memoria Dinamica [New & Delet]

```cpp
#include<iostream>
void FuncionEjemplo()
{
    int     varLocal = 10;      // Variable local en la pila
    int*    varHeap = new int;  // Asignación dinámica en el montón
    *varHeap = 20;
    std::cout << "Variable en la pila: " << varLocal << std::endl;
    std::cout << "Variable en el montón: " << *varHeap << std::endl;

    delete varHeap; // Libera la memoria del montón
}

int main()
{
    FuncionEjemplo();
    return 0;
}
```

EJERCICIOS

Ejercicios

01

Escriba un programa que pida al usuario que escriba dos números, obtenga esos dos números del usuario e imprima la suma, producto, diferencia y cociente de los dos números. [**NOTA:** Validar Ingresos y administrar manualmente la memoria]

02

Escriba un programa que pida al usuario que escriba dos enteros, obtenga los números del usuario y luego imprima el número más grande, seguido de las palabras "es más grande". Si los números son iguales, imprima el mensaje "Estos números son iguales". [**NOTA:** Validar Ingresos y administrar manualmente la memoria]

03

Escriba un programa que lea el radio de un círculo como un entero e imprima el diámetro del círculo, la circunferencia y el área. Use el valor constante 3.14159 para pi. Realice todos los cálculos en instrucciones de salida.
[**NOTA:** Validar Ingresos y administrar manualmente la memoria]

04

Escriba un programa que imprima un cuadro, un óvalo, una flecha y un diamante como se indica a continuación:

[**NOTA:** Validar Ingresos y administrar manualmente la memoria]

05

Escriba un programa que lea dos enteros, determine e imprima si el primero es múltiplo del segundo.
[**NOTA:** Validar Ingresos y administrar manualmente la memoria]