Ayudantia 1

Carlos Lagos - carlos.lagosc@usm.cl

C++

Estructura básica

```cpp
#include <iostream>

using namespace std;

int main(){
    int n;
    cin >> n;

    cout << "Hola mundo " << n << endl;

    return 0;
}
```

Variables y tipos de datos

En el ámbito de la programación, una variable se define como un espacio de almacenamiento identificado por un nombre simbólico, que guarda un valor y está asignado a una ubicación en la memoria del sistema. Por otro lado, un tipo de dato determina cómo se pretende emplear una variable, proporcionando al compilador o intérprete indicaciones sobre su uso.

Variables y tipos de datos

En C++, los tipos de datos primitivos incluyen:

1. **Enteros**: como `int`, `short`, `long`, y `long long`.
2. **Punto flotante**: como `float` y `double`.
3. **Caracteres**: como `char`.
4. **Booleano**: como `bool`.
5. **Punteros**: como `int*`, `char*`, etc.

Variables y tipos de datos

```cpp
#include <iostream>

using namespace std;

int main(){
    int numero4bytes = 123;
    long long numero8bytes = 1323132;
    bool valorVoF1bytes = false;
    float decimal4bytes = 0.5;
    double decimalconmaspres8bytes = 0.1
    // sizeof(tipo)
    return 0;
}
```

Operadores en C++

Operadores Aritméticos:

1. **Suma (`+`)**: Se utiliza para sumar dos valores.
2. **Resta (`-`)**: Resta el valor del operando derecho del valor del operando izquierdo.
3. **Multiplicación (`*`)**: Multiplica los dos operandos.
4. **División (`/`)**: Divide el operando izquierdo por el operando derecho.
5. **Módulo (`%`)**: Devuelve el resto de la división del operando izquierdo por el operando derecho.

Operadores de Comparación:

1. **Igualdad (`==`)**: Comprueba si dos valores son iguales.
2. **Desigualdad (`!=`)**: Comprueba si dos valores son diferentes.
3. **Mayor que (`>`)**: Comprueba si el valor del operando izquierdo es mayor que el del operando derecho.
4. **Menor que (`<`)**: Comprueba si el valor del operando izquierdo es menor que el del operando derecho.
5. **Mayor o igual que (`>=`)**: Comprueba si el valor del operando izquierdo es mayor o igual que el del operando derecho.
6. **Menor o igual que (`<=`)**: Comprueba si el valor del operando izquierdo es menor o igual que el del operando derecho.

Operadores

Operadores Lógicos:

1. **AND lógico (`&&`)**: Devuelve `true` si ambos operandos son `true`.
2. **OR lógico (`||`)**: Devuelve `true` si al menos uno de los operandos es `true`.
3. **NOT lógico (`!`)**: Invierte el valor de su operando.

Operadores de Asignación:

1. **Asignación (`=`)**: Asigna el valor del operando derecho al operando izquierdo.
2. **Asignación con suma (`+=`)**: Incrementa el valor del operando izquierdo por el valor del operando derecho.
3. **Asignación con resta (`-=`)**: Decrementa el valor del operando izquierdo por el valor del operando derecho.
4. **Asignación con multiplicación (`*=`)**: Multiplica el valor del operando izquierdo por el valor del operando derecho.
5. **Asignación con división (`/=`)**: Divide el valor del operando izquierdo por el valor del operando derecho.
6. **Asignación con módulo (`%=`)**: Asigna el módulo del operando izquierdo con el operando derecho.

Estructura de Control `if`

La estructura `if` se utiliza para ejecutar un bloque de código si se cumple una condición específica.

```
if (condición) {
    // Código a ejecutar si la condición es verdadera
}
```

Estructura de Control `else if`

La estructura `else if` se utiliza después de un `if` para evaluar múltiples condiciones secuenciales si la condición del `if` no se cumple.

```
if (condición1) {
    // Código a ejecutar si la condición1 es verdadera
} else if (condición2) {
    // Código a ejecutar si la condición2 es verdadera
} else {
    // Código a ejecutar si ninguna condición es verdadera
}
```

Estructura de Control `while`

La estructura `while` se utiliza para ejecutar un bloque de código repetidamente mientras se cumpla una condición específica.

```
while (condición) {
    // Código a ejecutar mientras la condición sea verdadera
}
```

Estructura de Control `do while`

La estructura `do while` es similar a `while`, pero garantiza que el bloque de código se ejecute al menos una vez, ya que evalúa la condición después de la ejecución del bloque.

```
do {
    // Código a ejecutar al menos una vez
} while (condición);
```

Estructura de Control `for`

La estructura `for` se utiliza para ejecutar un bloque de código un número específico de veces.

```
for (inicialización; condición; actualización) {
    // Código a ejecutar mientras la condición sea verdadera
}
```

La inicialización se realiza antes de que comience el bucle, la condición se evalúa antes de cada iteración y la actualización se ejecuta al final de cada iteración.

Declaración y Acceso a los Elementos

```
tipo_de_dato nombre_arreglo[tamaño];
nombre_arreglo[indice];
```

Por ejemplo:

```
int numeros[5];
int primer_numero = numeros[0];
```

Inicialización y Tamaño

```
tipo_de_dato nombre_arreglo[tamaño] = {valor1, valor2, ..., valorN};
int TAMANO_ARREGLO = 5;
int numeros[TAMANO_ARREGLO];
```

Por ejemplo:

```
int numeros[5] = {1, 2, 3, 4, 5};
```

Iteración y Limitaciones

```
for (int i = 0; i < TAMANO_ARREGLO; i++) {
    // Acceder a elementos usando el índice i
}
```

- Tamaño fijo.
- Sin comprobación de límites.

Los arreglos en C++ son fundamentales para almacenar y manipular datos de manera secuencial.

Arreglos

```cpp
#include <iostream>

using namespace std;

int main(){
    int numeros[] = {10,7,2,6,3};
    for(int i = 0; i < 5; i++){
        if(numeros[i] % 2 == 0){
            cout << *(numeros + i) << endl;
        }else{
            cout << numeros[i]*-1 << endl;
        }
    }
    return 0;
}
```

Arreglo 2D

```cpp
#include <iostream>

using namespace std;

int main(){
    int arreglo2d[2][3] = {
        {5,6,1},
        {-1,-1,-1}
    };
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 3; j++){
            cout << arreglo2d[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;

    return 0;
}
```

Structs en C++

En C++, un `struct` es una forma de definir un nuevo tipo de datos que puede contener diferentes tipos de datos agrupados bajo un solo nombre.

Declaración de un Struct

```
struct NombreStruct {
    tipo_de_dato1 mi_dato1;
    tipo_de_dato2 mi_dato2;
    // ...
};
```

Por ejemplo:

```
struct Persona {
    std::string nombre;
    int edad;
};
```

Uso de un Struct

Una vez que se ha definido un `struct`, se puede utilizar para crear variables como cualquier otro tipo de datos.

```
NombreStruct variable_struct;
variable_struct.mi_dato1 = valor;
variable_struct.mi_dato2 = otro_valor;
```

Por ejemplo:

```
Persona persona1;
persona1.nombre = "Juan";
persona1.edad = 30;
```

Punteros en C++

En C++, los punteros son variables que almacenan direcciones de memoria como su valor. Son muy útiles para manipular datos y estructuras de manera dinámica.

Declaración de Punteros

Un puntero se declara indicando el tipo de dato al que apunta, seguido del operador de asterisco `*` y el nombre del puntero.

```
tipo_de_dato *nombre_puntero;
```

Por ejemplo:

```
int *ptr_entero;
```

Asignación de Direcciones de Memoria

Los punteros se pueden inicializar con la dirección de memoria de una variable utilizando el operador de dirección `&`.

```
int variable = 10;
int *ptr_variable = &variable;
```

Acceso al Valor Apuntado

Para acceder al valor al que apunta un puntero, se utiliza el operador de indirección `*`.

```
int valor = *ptr_variable;
```

Esto asignará el valor de la variable apuntada por `ptr_variable` a la variable `valor`.

Uso de Punteros

Los punteros son utilizados en una variedad de situaciones, incluyendo la manipulación de arreglos dinámicos, la asignación de memoria dinámica, y la implementación de estructuras de datos avanzadas.

```
int main() {
    int variable = 10;
    int *ptr_variable = &variable;
    int valor = *ptr_variable;
    return 0;
}
```

Ejercicios

Plaza del Teatro

La Plaza del Teatro en la capital de Berland tiene una forma rectangular con tamaño $n \times m$ metros. Con ocasión del aniversario de la ciudad, se decidió pavimentar la Plaza con losas de granito cuadradas. Cada losa tiene un tamaño de $a \times a$.

¿Cuál es el menor número de losas necesarias para pavimentar la Plaza? Se permite cubrir una superficie mayor que la de la Plaza, pero la Plaza debe estar cubierta. No se permite romper las losas. Los lados de las losas deben ser paralelos a los lados de la Plaza.

Plaza del Teatro

Formato de entrada

La entrada contiene tres números enteros positivos en la primera línea: n, m y a ($1 \leq n, m, a \leq 10^9$).

Formato de salida

Escribe el número necesario de losas.

Plaza del Teatro

Ejemplo

Entrada	Salida
6 6 4	4

Subcadenas Pares

Se te da una cadena $s = s_1 s_2 \dots s_n$ de longitud n, que solo contiene dígitos del 1 al 9.

Una subcadena $s[l \dots r]$ de s es una cadena $s_l s_{l+1} s_{l+2} \dots s_r$. Una subcadena $s[l \dots r]$ de s se llama **par** si el número representado por ella es par.

Encuentra la cantidad de subcadenas pares de s. Ten en cuenta que, aunque algunas subcadenas sean iguales como cadenas, si tienen diferentes l y r, se cuentan como subcadenas diferentes.

Subcadenas Pares

Formato de entrada

La primera línea contiene un entero n $(1 \leq n \leq 65000)$ — la longitud de la cadena s.

La segunda línea contiene una cadena s de longitud n. La cadena s está compuesta únicamente por los dígitos 1, 2, ..., 9.

Formato de salida

Imprime el número de subcadenas pares de s.

Subcadenas Pares

Ejemplo 1

Entrada	Salida
4 1234	6

Ejemplo 2

Entrada	Salida
4 2244	10

Vectores Perpendiculares

Se te dan dos vectores en dos dimensiones, $\mathbf{a} = (a_1, a_2)$ y $\mathbf{b} = (b_1, b_2)$. Queremos saber si los vectores tienen un ángulo de 90 grados entre ellos.

Vectores Perpendiculares

Formato de entrada

La primera línea contiene dos enteros a_1 y a_2 ($-10^6 \leq a_1, a_2 \leq 10^6$) — las coordenadas del primer vector $\mathbf{a}$.

La segunda línea contiene dos enteros b_1 y b_2 ($-10^6 \leq b_1, b_2 \leq 10^6$) — las coordenadas del segundo vector $\mathbf{b}$.

Formato de salida

Imprime "Si" si los vectores forman un ángulo de 90 grados entre sí. Imprime "No" en caso contrario.

Vectores Perpendiculares

Requisitos

- Usa struct para representar los vectores.
- Implementa una función llamada verificar_perpendicular que reciba dos vectores y determine si son perpendiculares.

Vectores Perpendiculares

Ejemplo 1

Entrada	Salida
1 2 -2 1	Si

Ejemplo 2

Entrada	Salida
1 1 2 -2	No

Vectores Perpendiculares

HINT

Para verificar si dos vectores $\vec{u}$ y $\vec{v}$ son perpendiculares, podemos utilizar el producto punto entre ellos. Según la ecuación:

$$\vec{u} \cdot \vec{v} = |\vec{u}| \cdot |\vec{v}| \cdot \cos\theta$$

donde θ es el ángulo entre los vectores. Si $\vec{u} \cdot \vec{v} = 0$, entonces los vectores son perpendiculares.