PROGRAMACION:

Simbología:

// ⟶ comentamos.
string ⟶ cadena de texto (palabras).
pow ⟶ (x,y) = x^y
sqrt ⟶ $\sqrt{x}$ ⟶ raiz cuadrada
Sin ⟶ seno de (x)
tan ⟶ tangente de (x)
cos ⟶ coseno de (x)
float ⟶ decimales
bool ⟶ boliano (SI O NO) (0 y 1)
double ⟶ nº mas grandes
int ⟶ variable numeros enteros.
endl ⟶ saltar un espacio (acompañado de un ;)
Cout ⟶ escribir mensaje (<<)
Cin ⟶ guardar variable (>>)
% ⟶ modulo (resto de : entre dos numeros).
== ⟶ igualdad
&& ⟶ "y"
|| ⟶ "o"
! ⟶ distinto/negacion
a++ ⟶ a+1
/ ⟶ división
* ⟶ multiplicacion
<, ≤; ≥, > ⟶ mayor, menor, mayor o igual, menor o igual.

UNCIONES en C++:

declaración de una función:

```
Tipo nombre funcion ( [tipo nombreargumento], [tipo nombreargumento])
{
   return valor;
}
```

(una función siempre retorna algo, por tanto es obligatorio declararle un tipo, darle un nombre a dicha función, (para poder identificarla y llamarla durante la ejecución), y despues al interior podemos poner los parametros).

Ejemplo 1:

```
int cubo (int n){
int val = n*n*n;
return val;
}
int main(){
cout << cubo(2);
}
```

retorna variable

deben ser del mismo tipo.

Ejemplo 2:

Función dentro de otra:

```
int cubo (int n){
int val = n*n*n;
return val;
}
int volumen (int lado){
return cubo(lado);
}
int main (){
cout <<"Ingrese el ancho del cubo:";
int a;
cin>>a;
cout<<"El volumen es": << volumen (a
}
```

ARREGLOS EN C++:

Un arreglo es un conjunto finito de elementos tambien llamado vector.
El nombre de la variable del arreglo se le conoce como identificador y al numero de elementos como rango o largo.

tipo de elemento identificador [largo];
int a[10];
(se forma un arreglo de 10 elementos llamado a;
[ ] ⟶ va de 0 a n-1).

int a = 5 (donde a es una variable de tipo int y valor 5).
int a[10] = (donde a es un arreglo de largo 10).
int a [5] = 3; ⟶ (la posicion del arreglo 5 vale 3).
int b = a[5] ⟶ (la variable b, tiene el mismo valor que la pocsicion 5).
a [1] = a[5] · a[9] ⟶ (arreglo en 1, vale la multiplicacion de los arreglos en la posicion 5 y 9).

- Recorrer arreglos:

```
int i;
for (i=0; i<10; i++) // largo del arreglo.
a[i] = i;
}
```

- Sumar arreglos:

```
int i;
int suma = 0;
for (i=0; i<10; i++){
suma += a[i];
}
```

- mayor de un arreglo:

```
int i;
int mayor = a[0];
for (i=1; i<10; i++){
if( a[i] > mayor){
mayor = a[i];
}
}
```

- Buscar en un arreglo:

```
int i;
int posicion = -1;
int buscado = 10;
for(i=0; i<10; i++){
if(a[i] == buscado){
posicion = i;
}
}
```

UNTEROS:

```
int a=5;
cout << &a;
```

↑ puntero.

(0x7fff1a62c0ec)
(esto es el lugar de memoña)

```
int *p;
int a=5;
p=&a;
cout<<p;
```

cambiar el valor de memoña:

```
int a=5;
int *b = &a;
*b = *b+2;
cout<<a; //imprime 7.
```

cambiar en funcion:

```
void f(int *c){
*c = *c+2;
}
int main()
{
 fnta=5;
 f(&a);
cout<<a;
return 0;
}
```

ALMACENAR DIR MEMORIA

```
int main(){
int num,*dir_num;
 num=20;
dir_num = &num;
cout << "Numero:" << *dir_num << endl;
cout << "Direccion de mem:" << dir_num << endl;
```

MATRICES EN C++:

(Arreglos multidimensionales)

```
int [10][3];
```

- recorrer una matriz:

```
int i;
for(i=0; i<10; i++){
for (j=0; j<10; i++){
}
}
```

- numero rand:

```
int i,j;
for(i=0; i<10; i++){
for(j=0; j<3; j++){
  a[i][j] = rand()%10;
}
}
```

- matriz 5x5 suma de diagonal

```
int m[5][5];
int suma=0;
for(int i=0, j<5; j++){
for(int j=0; j<5; j++){
 if(i==j){
 suma+=m[i][j]
 }
 }
 }
```

- mayor de un arreglo:

```
int i,j, int mayor=0;
for(i=0; i<10; i++){
for(j=0; j<3; j++){
If(a[i][j]>mayor){
  mayor = a[i][j];
  }
  }
  }
```

- Buscar una matriz:

```
int buscado=9
int x=-1; y=-1;
for(i=0; i<10; i++){
for(j=0; j<3; j++){
If(a[i][j]==boscado){
  x=i; y=i;
}}}
```

- eliminar triangulo superior (metodo burbuja).

```
int m[5][5];
for(int i=0; i<5; i++){
for(j=0; j<5; j++){
 If(i>j){
 m[i][j]=0
}
}
}
  aux= m[i][j];
  m[i][j] = m[j][i];
  m[j][i] = aux;
```

Introducción a la OOP:

Programación orientada a objeto:

→ forma de programar basada en la vida real, Todo es un objeto y posee cualidades, se pueden reutilizar, e incluso, un objeto puede interactuar con otros objetos.

Clase:

→ molde para objetos / sustantivo común
→ se declara usando class.

atributos:

→ variables que posee una clase.

constructor:

→ función" y sirve para definir el estado inicial de un objeto.
→ tienen el mismo nombre que la clase.
→ no tiene retorno.

metodos:

→ acción ejercida por un metodo es invocar.

string getNombre { return nombre; }.

void setNombre { this → nombre = nombre; }.
(string nombre)

Objeto:

→ particularización de una clase (sustantivo propio o comun).

Privado: solo se acceden desde dentro de la clase

Público: pueden ser accedidos desde fuera

Ejemplo clase:

```
#include <iostream>
Using namespace std
Class Perro {
Private:
    string nombre;
    String raza;
    int edad;
Public:
Perro (string nombre, string raza, int edad) {
 this → nombre = nombre;
 this → raza = raza;
 this → edad = edad;
}
string getnombre { return nombre; }
string getraza { return raza; }
int getedad { return edad; }.
void setnombre (string nombre) {
    this → nombre = nombe;
}
```

...de un objeto. (Va en el main).

```
* perro1 = new Perro("nombre", "raza", "edad");
```

nombre clase — atributos clase

→ puntero de nuevo objeto.

usar sus métodos a través de ⟶ sus métodos.

Ejemplo:

```
int main () {
Perro *perro1 = new Perro("cachupin", "quiltro", 3);
cout << "El perro se llama" << perro1->getNombre() << endl;
cout << "El perro tiene " << perro1->getedad() << "años de edad" << endl
return 0;
```

Arreglo:

- matriz o vector es una zona de almacenamiento continuo que contiene una serie de elementos de un mismo tipo.
  - `nombreclase * nombrearreglo [indice];`
    `Alumno * curso [2];`
  - como ingresar un objeto al arreglo.

    por asignación: creamos el objeto y lo asignamos al arreglo en la misma línea

    `curso[i] = new Alumno (atributos).`

    paso por valor: objeto creado en otra variable y luego lo asignamos al arreglo.

    ```
    alumno * temp = new alumno (atributos)
    Curso[i] = temp;
    ```
  - acceder a métodos

    `Curso[i] ⟶ getNombre();`

ón entre objetos (objetos como parametros).

→ forma:

<tipo dato> <nombremetodo> (<nombe clase> * <objeto creado>);

Ejemplo 1:

```
void ladrar (Gato *gatito) {
}
```

(asumiendo que clase gato posee un metodo ladrar)

Ejemplo 2:

```
Class Gato {
    Private:
        string nombe;
    Public:
      Gato (string nombre){
       this→ nombre=nombre;
      }.
    string getNombre { return nombre;}.
 };

 Class Perro{
    Private:
        String raza;
    Public:
       Perro(string raza){
        this →raza=raza;
       }
    string ladrar (Gato *gato) {
       return ("Guau Guau Guau");
       }.
 int main ()
{
  Gato * Tom = new Gato ("Tom");
 Perro * bobby = new Perro ("Chihuahua");
cout << Bobby →ladrar (Tom) << endl;
}
```

s como atributos de una clase)

Ejemplo 1

```
Class motor : {
    Private:
        string modelo;
        int cilindrada;
    Public
     motor (string modelo, int cilindrada){
        this→modelo = modelo;
        this→cilindrada = cilindrada;
     }.
Class Auto: {
    Private:
        string marca;
        string modelo;
        Motor * motor;
    Public:
    Auto(string marca, string modelo, Motor * motor){
        this → marca = marca;
        this → modelo = modelo;
        this → motor * motor;
}
```

Introduccion a la herencia:

- crear una clase hija que tomara toda la informacion de una clase padre..
- Clase padre: clase que sirve de molde en comun para la creacion de clases derivadas. (instancia como clase normal, solo que cambia de privado a protected).
- Clase hija: clase derivada de una padre.

definicion clase hija:

Class <nombe clase hija> : <public> <nombre clase padre>

Constructor clase hija:

<clase hija>(atributos propios y padre con : <clase padre>(atributos padre sin su tipo) su tipo)

```
ces Cuenta

protected:
    int saldo;
    string nrtitular;
public:
 cuenta( int saldo, string nrtitular) {
    this →saldo=saldo;
    this→ nrtitular = nrtitular;
  }

        int getsaldo() { return saldo; }
        string getnrtitular() { return nrtitular; }
        void setdeposito (int monto) {
            saldo = saldo + monto;
        }
        };
```

clase padre.

```
Class CuentaRut: public Cuenta{
  Private:
      int costogiro;
  public:
   CuentaRut (string nrtitular, int saldo, int costogiro): Cuenta (saldo, nrtitular) {
        this → costogiro = 300;
    }
   int getcostogiro() { return costogiro; }.

void selgiro (int monto) {
      int maximo = saldo - costogiro;
   if( monto <= maximo) {
      saldo = saldo - (monto + costogiro);
   }
   else {
   }
 }
};
```

clase hijo.

rga y polimorfismo:

→ **sobrecarga**: posibilidad de tener dos o mas funciones con el mismo nombre pero funcionalidad diferente.
dos o mas funcones con el mismo nombre, pero hacen cosas diferentes.

Ejemplo:

```
Class Perro {
    Private:
        string nombre;
    Public:
    Perro(string nombre){
     this → nombre = nombre;
    }
string getNombre () { return nombre; }.
string ladrar (string persona) {
  return ("Guau Guau");
}
 string ladrar (){
  return ("Guauuuuuuuu");
}
```

se llaman igual, retornan distinto.

→ **polimorfismo**: a la propiedad por la que es posible enviar mensajes sintacticamente iguales a objetos de tipo distintos.