# **ارزیابی زبان سی پلاس پلاس**

## **خوانایی**

زبان برنامه‌نویسی سی پلاس پلاس به خوانایی بالا و ساختار ساده مشهور است. نحوه نوشتار و طراحی نحوی منطقی آن برای برنامه نویسان درک آسانی دارد. استفاده از اپراتورها و ویژگی‌های زبان مانند ارث‌بری و پلی‌مورفیسم باعث می‌شود که کد خوانا و تعمیم‌پذیر باشد.

### **سادگی**

زبان برنامه‌نویسی ++C یک زبان قدرتمند و گسترده است که از زبان C به‌طور مستقیم گسترش یافته است. این زبان از ویژگی‌های متعددی نظیر انعطاف‌پذیری، کارایی، و امکانات شیءگرایی و ... پشتیبانی می‌کند. اما در مقایسه با زبان‌های سطح بالا مانند Python یا JavaScript، این زبان معمولاً به‌عنوان یک زبان پیچیده‌تر شناخته می‌شود. اما در کنار این نقاط قوت ، ضغف‌هایی نیز دارد که سبب شده نسبت به زبان‌های سطح بالا مانند Python یا JavaScript، این زبان معمولاً به‌عنوان یک زبان پیچیده‌تر شناخته شود.

* :Object-Oriented این زبان شیءگرا است که این امکان را فراهم می‌کند که برنامه نویسان کد خود را با استفاده از اشیا سازماندهی کنند. این ویژگی باعث افزایش خوانایی و قابلیت توسعه کد می‌شود اما از طرفی نیز می‌تواند برنامه را به یک ساختار پیچیده‌تر تبدیل کنند و سادگی را کاهش دهد.
* قدرت و انعطاف بالا: این زبان به برنامه‌نویسان امکان مدیریت حافظه را می‌دهد و از ویژگی‌های پرکاربردی مانند ارث‌بری و پلی‌مورفیسم پشتیبانی می‌کند. این انعطاف باعث می‌شود که بتوان به راحتی با مسائل پیچیده و متنوع مواجه شد. اما مدیریت دستی حافظه ممکن است سطح پیچیدگی برنامه را افزایش دهند.
* پشتیبانی ازپارادایم‌های متعدد: نه تنها یک زبان شیءگرا است بلکه از پارادایم‌های برنامه‌نویسی متعددی مانند برنامه‌نویسی رویه‌ای نیز پشتیبانی می‌کند. این ویژگی به برنامه‌نویسان امکان انتخاب بهترین راه برنامه‌نویسی برای مسئله خود را می‌دهد. پس سادگی را افزایش می‌دهد.
* تعداد زیادی کتابخانه: ++C دارای تعداد زیادی کتابخانه است که می‌تواند برنامه را ساده‌تر یا پیچیده‌تر کند، به تبع آنکه از امکانات بیشتری برخوردار است.

در مثال زیر، از ارث‌بری و پلی‌مورفیسم استفاده شده است که از یک سو خوانایی کد را افزایش می‌دهد، اما از سوی دیگر ممکن است برای برنامه نویسان تازه وارد گیج‌کننده باشد.

#include <iostreamm>  
  
class Animal {  
public:  
    virtual void sound() {  
        std::cout << "Animal makes a sound" << std::endl;  
    }  
};  
  
class Dog : public Animal {  
public:  
    void sound() override {  
        std::cout << "Dog barks" << std::endl;  
    }  
};  
  
int main() {  
    Animal\* myAnimal = new Dog();  
    myAnimal->sound();  
  
    delete myAnimal;  
    return 0;  
}

به طور کلی، اگرچه C++ با استفاده از امکانات و ویژگی‌های خود امکانات بالایی را به برنامه‌نویسان ارائه می‌دهد، این امکانات به همراه باعث افزایش پیچیدگی و تعداد خطوط کد می‌شوند. اما با مهارت و تجربه، برنامه‌نویسان می‌توانند از این امکانات به بهترین شکل استفاده کنند و کدهای خواناتر و قابل نگهداری تر ایجاد کنند.

### **تعامد**

زبان C++ به عنوان یک زبان با تعامد پایین شناخته می‌شود. که در ادامه بعضی دلایل ان را بررسی میکنیم.

* روش‌های متعدد برای دستیابی به اهداف مشابه: به طور معمول چندین روش برای انجام کارهای یکسان یا مشابه فراهم می‌کند. به عنوان مثال، روش‌های متعددی برای اعلان و مقداردهی متغیرها و ساختارهای متفاوت برای حلقه‌ها یا تصمیم‌گیری‌ها وجود دارد.

int x = 5;

auto y = 5;

auto square = [](int x) { return x \* x; }; // تعریف تابع با استفاده از عبارت لامبدا

int power(int base, int exponent){ // تعریف تابع با استفاده از تابع معمولی

int result = 1;

for (int i = 0; i < exponent; ++i) {

result \*= base;

}

return result;

}

inline double cube(double x) { // تعریف تابع با استفاده از تابع inline

return x \* x \* x;

}

* نحو متفاوت برای مفاهیم مشابه: برخی از ساختارهای زبان برای مفاهیم مشابه از نحو متفاوتی استفاده می‌کنند، که می‌تواند منجر به پیچیدگی در فهم برنامه‌نویسان شود. به عنوان مثال، نحوه استفاده از تفاوت‌هایی بین روش‌های تعریف متغیرها وجود دارد.

struct Point{

int x;

int y;

;{

Point origin {0, 0}=;

Point\* ptr = &origin;

int xCoordinate1 = (\*ptr).x

int xCoordinate2 =ptr->x;

int xCoordinate3 = origin.x;

* در زبان C++، می‌توانیم یک شیء را از هر نوع اصلی به استثنای void تعریف کنیم، اما این قاعده برای تمپلیت ها صادق نیست. به عبارت دیگر، در تعریف قالب زیر، برای اینکه کد کار کند، می‌بایست به صورت صریح نوع T را به void تبدیل کنیم:

template<typename T>

T f} ()

// code

{

T = void;

* یک تابع میتواند هر نوع داده ای را بازگرداند بجز ارایه و برای ان باید از اشاره‌گر کمک بگیرد.

اگرچه این جنبه‌ها به کم بودن تعامد زبان C++ اشاره دارند، این ویژگی‌ها همچنان امکانات بسیاری را برای برنامه‌نویسان فراهم می‌کنند. تعیین اینکه این ویژگی‌ها چقدر به عنوان یک مزیت یا معایب در توسعه نرم‌افزار شناخته می‌شوند، بستگی به نیازها و ترجیحات برنامه‌نویسان دارد. همچنین موارد بالا دلیل بر نداشتن تعامد نیست در ادامه به یک مثال از ترکیب عناصر متفاوت میپردازیم. در این مثال، Orthogonality از طریق ترکیب دو کلاس Shape و Color در کلاس Circle نشان داده شده است. این می‌تواند به یک پروژه امکان ترکیب مفاهیم مختلف را بدهد، اما همچنین ممکن است باعث پیچیدگی بیش از حد در کد شود.

#include <iostream>  
  
class Shape {  
public:  
    virtual void draw() {  
        std::cout << "Drawing a shape" << std::endl;  
    }  
};  
  
class Color {  
public:  
    virtual void fill() {  
        std::cout << "Filling with color" << std::endl;  
    }  
};  
  
class Circle : public Shape, public Color {  
public:  
    void draw() override {  
        std::cout << "Drawing a circle" << std::endl;  
    }  
  
    void fill() override {  
        std::cout << "Filling circle with color" << std::endl;  
    }  
};  
  
int main() {  
    Circle myCircle;  
    myCircle.draw();  
    myCircle.fill();  
  
    return 0;  
}

### **نوع داده**

زبان C++ یک زبان برنامه‌نویسی گسترده و انعطاف‌پذیر است و از نظر تئوری، تقریباً هر نوع داده‌ای را می‌توانید در C++ ایجاد کنید. نوع‌های داده استاندارد C++ شامل موارد زیر است.

1.اعداد صحیح (Integers):

* + Int: نمایانگر اعداد صحیح است. بسته به سیستم ممکن است اندازه آن تغییر کند (مثلاً 32 یا 64 بیت).
  + short : نمایانگر اعداد کوتاهتر (معمولاً 16 بیت).
  + long : نمایانگر اعداد بلندتر (معمولاً حداقل 32 بیت).
  + long long : نمایانگر اعداد بلندتر (ممکن است حداقل 64 بیت باشد).

2.اعداد حقیقی (Floating-Point Numbers):

* + float : نمایانگر اعداد حقیقی با اعشار (32 بیت).
  + Double: نمایانگر اعداد حقیقی با اعشار (حداقل 64 بیت).
  + long double : نمایانگر اعداد حقیقی با اعشار بیشتر از double (اندازه ممکن از سیستم به سیستم متفاوت باشد.)

3.اعداد مختلط(Complex Numbers):

* + std::complex<T> : نمایانگر اعداد مختلط با اعشار (T)میتواند هر نوع داده از دسته دوم باشد.

4.کاراکترها (Characters):

* + char : نمایانگر یک کاراکتر (8 بیت).
  + wchar\_t : نمایانگر یک کاراکتر ویژه (حداقل 16 بیت).

5.بولی (Boolean):

* + bool : نمایانگر مقادیر منطقی true یا false.

6.پوینترها (Pointers):

* + \*T

7.مرجع (Refrences):

* + &T

8.ماکروها (Macros):

* + #define : عمدتاً برای جایگزینی کد متن با کد متن دیگر استفاده میشود.

9.آرایه‌ها (Arrays):

* + T array[size]

10.ساختارها (Structures):

* + struct : تعریف یک ساختار از چند متغیر با نوع‌های مختلف.

11.اشیاء (Objects):

* + Class : با استفاده از Object-Oriented Programming

12.نوع void:

* + معمولاً برای نمایانگر عدم بازگشت از یک تابع استفاده می‌شود

همچنین با استفاده از کتابخانه ی STL میتوان استک، لیست، مپ، وکتور، ست و ... داشت. تعداد فراوان نوع داده و امکانات زبان C++ در مقایسه با بسیاری از زبان‌های برنامه‌نویسی، از جمله زبان‌های C، Java، Python، و...، به آن انعطاف بیشتری داده است.

### **طراحی نحوی**

سی پلاس پلاس به لحاظ سینتکس و نحوه نوشتار، سادگی بالایی دارد. استفاده از شیءگرایی، اپراتورهای مناسب، و قوانین مشخص در زبان باعث ایجاد کدی قابل فهم می‌شود. با این حال این سینتکس گاهی به عنوان یک نقطه ضعف مطرح می‌شود. برخی از ساختارهای زبان C++ ممکن است برنامه نویسان جدید را گیج کند. به عنوان مثال استفاده از نمادهایی خاص مانند اشاره‌گرها میتواند پیچیدگی افزایش دهد.