دانشگاه اصفهان
دانشکده مهندسی کامپیوتر

گزارش فاز اول پروژه زبان‌های برنامه نویسی

زبان برنامه نویسی C++

تهیه کنندگان:
متین اعظمی
پوریا طلائی
عسل خائف

استاد درس:
آقای دکتر آرش شفیعی

نیم‌سال اول ۴۰۳ - ۴۰۴

فهرست مطالب

فهرست جداول

فهرست تصاویر

فصل ۱

مقدمه

۱ - ۱ - تاریخچه زبان C++

- آغاز و ابداع زبان C++: زبان C++ توسط بیارنه استراستروپ (Bjarne Stroustrup) در اوایل دهه ۱۹۸۰ در Bell Labs شرکت (AT-T) توسعه داده شد. این زبان در ابتدا به‌عنوان یک نسخه ارتقاء یافته از زبان C طراحی شد که ویژگی‌های شی‌گرا به آن افزوده می‌شد. به‌ویژه هدف آن این بود که برنامه‌نویسان قادر به نوشتن برنامه‌های پیچیده‌تر با ویژگی‌های شی‌گرا باشند، در حالی که هنوز از کارایی بالا و قابلیت‌های زبان C بهره‌مند باشند.
- هدف اولیه: C++ ابتدا به‌منظور ایجاد یک زبان برنامه‌نویسی با پشتیبانی از برنامه‌نویسی شی‌گرا (OOP) در کنار قابلیت‌های سطح پایین زبان C طراحی شد. ویژگی‌های OOP مانند ارث‌بری (inheritance)، چندریختی (polymorphism) و کپسوله‌سازی (encapsulation) به این زبان اضافه شدند تا برنامه‌نویسان قادر باشند کدهای پیچیده‌تر و قابل نگهداری‌تری بنویسند.
- نام‌گذاری C++: نام C++ به دلیل افزوده شدن ویژگی‌های جدید به زبان C انتخاب شد. علامت ++ به‌طور نمادین به افزایش یا ارتقای زبان C اشاره دارد.

۱ - ۲ - کاربردهای زبان C++

- سیستم‌های نرم‌افزاری پیچیده: C++ از ابتدا برای نوشتن سیستم‌های پیچیده و نرم‌افزارهای کاربردی طراحی شد که نیاز به سرعت بالا و دسترسی مستقیم به سخت‌افزار دارند. از این رو در سیستم‌عامل‌ها مانند (ویندوز و لینوکس) ، نرم‌افزارهای سیستمی و نرم‌افزارهای Embedded به‌طور گسترده‌ای استفاده می‌شود.
- توسعه بازی‌ها: C++ زبان اصلی برای توسعه بازی‌های کامپیوتری و گرافیکی است. موتورهای بازی‌سازی بزرگی مانند Unreal Engineاز C++ استفاده می‌کنند. این زبان به دلیل کارایی بالا و پشتیبانی از برنامه‌نویسی شی‌گرا برای توسعه بازی‌های پیچیده بسیار مناسب است.
- برنامه‌نویسی علمی و مهندسی: C++ در زمینه‌هایی مانند شبیه‌سازی‌های علمی، پردازش تصویر، پردازش

داده‌های بزرگ و مدل‌سازی فیزیکی استفاده می‌شود. به‌ویژه در حوزه‌های مهندسی و علوم کامپیوتر به دلیل قدرت پردازشی بالا و مدیریت دقیق حافظه کاربرد زیادی دارد.

- نرم‌افزارهای مالی: به دلیل سرعت و کارایی بالای C++، این زبان در توسعه نرم‌افزارهای مالی، تحلیل داده‌های بورس و مدیریت تراکنش‌های بانکی نیز کاربرد دارد.

۱-۳- هدف اصلی از طراحی C++

- رفع مشکلات زبان C : C++ به‌عنوان یک ارتقاء بر زبان C طراحی شد. یکی از مشکلات زبان C عدم پشتیبانی از ویژگی‌های شی‌گرا بود که در برنامه‌های پیچیده کارایی و نگهداری کد را دشوار می‌کرد. C++ این قابلیت‌ها را به زبان اضافه کرد، در حالی که همچنان از ساختارهای سطح پایین و کارایی بالای C بهره می‌برد.

- افزایش قدرت و انعطاف‌پذیری: C++ از همان ابتدا قصد داشت تا قدرت و انعطاف‌پذیری بیشتری را به برنامه‌نویسان بدهد. به‌ویژه با استفاده از ویژگی‌های شی‌گرا، کدهای پیچیده‌تر و انعطاف‌پذیرتری می‌توان نوشت.

- پشتیبانی از برنامه‌نویسی شی‌گرا: یکی از اصلی‌ترین اهداف C++ این بود که ویژگی‌های شی‌گرا را به زبان C اضافه کند، به‌طوری که برنامه‌نویسان بتوانند از ارث‌بری، چندریختی و کپسوله‌سازی برای نوشتن نرم‌افزارهای مقیاس‌پذیرتر و قابل نگهداری‌تر استفاده کنند.

۱-۴- مشکلات اولیه زبان C++

- پیچیدگی: یکی از مشکلات ابتدایی C++ پیچیدگی یادگیری آن بود. بسیاری از برنامه‌نویسان جدید با مفاهیم پیچیده‌ای مانند اشاره‌گرها، مدیریت حافظه دستی و ویژگی‌های شی‌گرا مواجه می‌شدند.

- مدیریت حافظه: اگرچه C++ به برنامه‌نویسان کنترل دقیقی بر حافظه می‌دهد، اما این امر باعث می‌شود که مدیریت حافظه به‌صورت دستی بسیار دشوار و مستعد خطا باشد. برای مثال، دسترسی به حافظه اشتباه یا فراموش کردن آزادسازی حافظه می‌تواند باعث ایجاد اشکالاتی مانند "Memory Leaks" و "Segmentation Faults" شود.

- عدم تطابق با زبان‌های سطح بالا: در ابتدا، بسیاری از برنامه‌نویسان سعی می‌کردند تا C++ را مانند زبان‌های سطح بالاتر استفاده کنند، اما این امر به‌خاطر پیچیدگی‌های خاص C++و نیاز به توجه بیشتر به جزئیات سخت‌افزاری ممکن نبود.

برای ارزیابی زبان C++ در مقایسه با زبان‌های دیگر و به‌ویژه زبان‌هایی که ویژگی‌های مشابه دارند، باید معیارهای مختلفی از جمله خوانایی، قابلیت اطمینان، کارایی، هزینه یادگیری و بهره‌وری، و قابلیت جابجایی را در نظر بگیریم. در اینجا یک تحلیل جامع از C++ در مقایسه با زبان‌های مشابه (مانند C ، Java، Python) ارائه می‌شود:

۱-۵- ویژگی‌های خاص C++ که آن را از زبان‌های مشابه متمایز می‌کند

- کنترل دقیق بر حافظه: یکی از بزرگترین ویژگی‌های تمایز C++ نسبت به زبان‌های مشابه، قابلیت کنترل دقیق بر حافظه است. در زبان‌هایی مانند C و C++، برنامه‌نویس باید به‌صورت دستی حافظه را تخصیص دهد و آن را آزاد کند. این ویژگی به زبان‌های سطح پایین‌تر این امکان را می‌دهد که از عملکرد بسیار بالا و بهینه استفاده کنند، به‌ویژه در سیستم‌های embedded و بازی‌ها. این ویژگی در زبان‌هایی مانند Java و Python وجود ندارد، زیرا این زبان‌ها از جمع‌آوری زباله (garbage collection) برای مدیریت حافظه استفاده می‌کنند.

- شی‌گرایی و چندریختی : C++ از اولین زبان‌هایی بود که پشتیبانی از ویژگی‌های شی‌گرایی را به زبان‌های سطح پایین اضافه کرد. این ویژگی در مقایسه با زبان‌هایی مثل C که شی‌گرایی ندارند، یک مزیت بزرگ به‌شمار می‌آید. به علاوه، C++ از چندریختی (polymorphism) و وراثت (inheritance) به‌خوبی پشتیبانی می‌کند که این امر نوشتن کدهای پیچیده و قابل نگهداری را ساده‌تر می‌کند.

- توانایی ترکیب ویژگی‌های سطح پایین و بالا: C++یک زبان چندپارادایمی است که هم از برنامه‌نویسی شی‌گرا (OOP) و هم از ویژگی‌های سطح پایین مانند دسترسی مستقیم به حافظه، کار با پورت‌ها و سخت‌افزار پشتیبانی می‌کند. این ویژگی باعث می‌شود که C++برای توسعه نرم‌افزارهای سیستم و برنامه‌های پیچیده با نیاز به کارایی بالا ایده‌آل باشد.

- پشتیبانی از Template و Generic Programming: C++ دارای قابلیت‌های پیشرفته‌ای مانند Templates است که امکان برنامه‌نویسی جنریک را فراهم می‌کند. این ویژگی به برنامه‌نویسان این امکان را می‌دهد که کدهای بازتر و انعطاف‌پذیرتری بنویسند که برای انواع مختلف داده‌ها کار کند.

۱-۶- ارزیابی زبان C++ بر اساس معیارهای مختلف

۱-۶-۱- خوانایی (Readability)

- C++: به‌طور کلی، خوانایی C++ نسبت به زبان‌های سطح بالا مانند Python یا Java پایین‌تر است. دلیل این امر استفاده از ویژگی‌های پیچیده‌ای مانند اشاره‌گرها (pointers) ، چندپارادایم بودن زبان، و نیاز به مدیریت حافظه دستی است. این ویژگی‌ها ممکن است باعث پیچیدگی در فهم کد و اشکال‌زدایی آن شوند.

- Java/Python: این زبان‌ها به‌خاطر سادگی و ساختار واضح‌تر خود، خوانایی بیشتری دارند. در Python به‌ویژه با وجود سینتکس ساده‌تر و نداشتن ویژگی‌هایی مانند اشاره‌گر، کدها بسیار قابل فهم‌تر هستند.

۱-۶-۲- قابلیت اطمینان (Reliability)

- C++: یکی از نقاط ضعف C++در مقایسه با زبان‌هایی مانند Java ،خطراتی مانند Memory Leaksو Segmentation Faultsاست. زیرا C++به‌طور دستی حافظه را مدیریت

می‌کند و این می‌تواند منجر به مشکلاتی در صورت خطای برنامه‌نویس شود. با این حال، این ویژگی برای سیستم‌های پیچیده و بازی‌ها که نیاز به کارایی بالا دارند، بسیار مفید است.

- Java: Java با استفاده از garbage collection و مدیریت خودکار حافظه، قابلیت اطمینان بیشتری دارد و کمتر مستعد مشکلات ناشی از مدیریت حافظه است.
- Python: Python نیز مانند Java از garbage collection استفاده می‌کند و به همین دلیل بیشتر از C++ قابلیت اطمینان دارد، به‌ویژه در پروژه‌های بزرگتر که مدیریت حافظه مشکل‌ساز می‌شود.

۱-۶-۳- کارایی (Performance)

- C++: C++ یکی از سریع‌ترین زبان‌های برنامه‌نویسی است. به‌خاطر آنکه برنامه‌نویسان کنترل دقیقی بر حافظه دارند، می‌توانند به بهینه‌ترین شکل ممکن از منابع استفاده کنند. این زبان برای برنامه‌هایی که به کارایی بالا نیاز دارند) مثل بازی‌ها، سیستم‌عامل‌ها و برنامه‌های(real-time بسیار مناسب است.
- Java/Python: مقابل، زبان‌های سطح بالاتر مانند Java و Python معمولاً از سرعت پایین‌تری برخوردارند، زیرا خودکار حافظه را مدیریت می‌کنند و به همین دلیل نیاز به منابع بیشتری دارند. Python به‌ویژه به‌خاطر مفسر بودنش کندتر از C++ است.

۱-۶-۴- هزینه یادگیری و برنامه‌نویسی (Learning Curve and Development Costs)

- C++: یادگیری C++ می‌تواند چالش‌برانگیز باشد، به‌ویژه برای مبتدیان. مفاهیم پیچیده‌ای مانند اشاره‌گرها، مدیریت حافظه دستی، و ویژگی‌های شی‌گرایی نیازمند زمان و تلاش برای یادگیری و درک عمیق هستند. این زبان برای برنامه‌نویسان مبتدی و تازه‌کار ممکن است دشوار باشد.
- Java/Python: در مقایسه،Python خاطر سینتکس ساده‌اش بسیار سریع‌تر یاد گرفته می‌شود و برای برنامه‌نویسان مبتدی مناسب است. Java نیز اگرچه کمی پیچیده‌تر از Python است، اما از C++ ساده‌تر است و برای یادگیری و توسعه سریع‌تر از C++ است.

۱-۶-۵- هزینه اجرایی (Execution Cost and Efficiency)

- C++: یکی از نقاط قوت C++ این است که برنامه‌های نوشته شده با آن معمولاً از کمترین منابع سخت‌افزاری استفاده می‌کنند و سریع‌ترین عملکرد را دارند.
- Java/Python: در حالی که Java و Python به دلیل نیاز به ماشین مجازی یا مفسر و مدیریت حافظه خودکار، از نظر کارایی نسبت به C++ کندتر عمل می‌کنند .

۱-۶-۶- قابلیت جابجایی (Portability)

- C++: C++برنامه‌ها را به کد ماشین تبدیل می‌کند، به همین دلیل ممکن است برای پلتفرم‌های مختلف نیاز به کامپایل مجدد داشته باشد.
- Java: یکی از مزایای اصلی Java این است که برنامه‌های نوشته شده با آن از ویژگی "write once, run anywhere" برخوردار هستند. زیرا کد جاوا به بایت‌کد تبدیل شده و در Java Virtual Machine (JVM) اجرا می‌شود که این امکان را می‌دهد تا بدون تغییر کد بر روی هر پلتفرم قابل اجرا باشد.
- Python: Python نیز به‌خاطر پشتیبانی از پلتفرم‌های مختلف، از جمله ویندوز، لینوکس، و مک، دارای قابلیت جابجایی خوبی است.

۱-۶-۷- نتیجه‌گیری

C++از نظر کارایی و کنترل دقیق بر منابع بسیار قدرتمند است و در برنامه‌هایی که نیاز به بهینه‌سازی‌های پیچیده دارند، ایده‌آل است. برای برنامه‌هایی که نیاز به سادگی و سرعت توسعه دارند، زبان‌هایی مانند Python یا Java ممکن است گزینه‌های بهتری باشند. اگر به دنبال توسعه سیستم‌های پیچیده و مقیاس‌پذیر با قابلیت‌های پیشرفته مانند OOP و کنترل دقیق هستید، C++ انتخاب بسیار مناسبی است.

۱-۷- پیاده‌سازی زبان C++: کامپایلر یا مفسر؟

- زبان C++ به‌طور کامل به کد ماشین ترجمه می‌شود، که پس از آن مستقیماً توسط سیستم‌عامل و سخت‌افزار اجرا می‌شود. به این معنی که C++ یک زبان کامپایل‌شده است، نه یک زبان مفسر.
- در این فرآیند، ابتدا کد منبع C++ توسط کامپایلر ترجمه می‌شود به کدهای ماشین یا بایت‌کدهایی که مستقیماً قابل اجرا روی سیستم هدف باشند. این کامپایلرها مسئول تبدیل کدهای نوشته‌شده در C++ به فرم قابل اجرا هستند.

۱-۸- کامپایلرهای رایج برای زبان C++

در حال حاضر چندین کامپایلر برای زبان C++ وجود دارد که هر یک ویژگی‌های خاص خود را دارند. برخی از محبوب‌ترین کامپایلرها عبارتند از:

۱-۸-۱- GCC (GNU Compiler Collection)

توسعه‌دهنده: GNU (Free Software Foundation).
مزایا:

- **منبع باز:** GCC یک کامپایلر منبع‌باز است و در بیشتر سیستم‌های عامل لینوکس و یونیکس استفاده می‌شود.

- **پشتیبانی از استانداردهای جدید C++:** GCC به‌طور مداوم با ویژگی‌های جدید C++ همگام است و از اکثر استانداردهای جدید $C++$ از جمله $C++11$، $C++14$، $C++17$، و $C++20$ پشتیبانی می‌کند.
- **قابلیت‌های بهینه‌سازی:** GCC یکی از کامپایلرهای معروف برای بهینه‌سازی کد است که سرعت اجرای برنامه‌ها را بهبود می‌بخشد.
- **پشتیبانی از پلتفرم‌های مختلف:** GCC قابلیت کار بر روی سیستم‌های مختلف مانند لینوکس، مک، ویندوز از طریق Cygwin و MinGW را دارد.

معایب:

- در مقایسه با کامپایلرهای تجاری، ممکن است بعضی از ویژگی‌ها یا بهینه‌سازی‌ها در GCC کمتر دقیق یا بهینه باشند.

۱-۸-۲- Clang

توسعه‌دهنده: Apple Inc. با مشارکت پروژه‌های متن‌باز.
مزایا:

- **سرعت کامپایل بالا:** Clang به‌عنوان یک کامپایلر سریع شناخته می‌شود که سرعت کامپایل بالاتری نسبت به برخی از دیگر کامپایلرها دارد.
- **پیغام‌های خطای دقیق و مفصل:** یکی از ویژگی‌های برجسته Clang پیغام‌های خطای بسیار واضح و دقیق آن است که برای برنامه‌نویسان مبتدی و حرفه‌ای مفید است.
- **پشتیبانی از استانداردهای جدید:** Clang همچنین از استانداردهای جدید C++ پشتیبانی می‌کند.
- **پشتیبانی از پلتفرم‌های مختلف:** مانند GCC، Clangنیز قابلیت اجرا بر روی پلتفرم‌های مختلف را دارد.
- **یکپارچگی با ابزارهای Apple:** به‌ویژه در محیط‌های macOS و iOS، Clang کامپایلر پیش‌فرض است.

معایب:

- برخی از ویژگی‌های خاص بهینه‌سازی Clang ممکن است نسبت به GCC کمتر پخته باشد.

۱-۸-۳- Microsoft Visual C++ (MSVC)

توسعه‌دهنده: Microsoft.
مزایا:

- **یکپارچگی با ویژوال استودیو:** MSVC به‌طور کامل با محیط توسعه‌ی Visual Studio که یکی از محبوب‌ترین IDE ها است، یکپارچه شده است. این یکپارچگی به برنامه‌نویسان C++ این امکان را می‌دهد که به راحتی برنامه‌های C++ را در ویندوز توسعه دهند.

- **ابزارهای پشتیبانی قوی:** MSVC ابزارهای زیادی برای اشکال‌زدایی و بهینه‌سازی کدها ارائه می‌دهد که برای توسعه نرم‌افزارهای ویندوزی بسیار مفید است.
- **بهینه‌سازی برای ویندوز:** MSVC برای بهینه‌سازی کدهایی که روی پلتفرم ویندوز اجرا می‌شوند، بسیار مناسب است.

معایب:

- MSVC معمولاً در مقایسه با GCC یا Clang پشتیبانی کمتری از استانداردهای جدید C++ خصوصاً $C++20$ دارد.
- **محدودیت‌های پلتفرمی:** MSVC عمدتاً برای ویندوز است و برای سیستم‌های عامل دیگر (لینوکس و مک) مناسب نیست.

۱-۸-۴- Intel C++ Compiler (ICC)

توسعه‌دهنده: Intel.
مزایا:

- **بهینه‌سازی‌های سطح پایین برای سخت‌افزارهای Intel:** ICC برای برنامه‌هایی که روی پردازنده‌های Intel اجرا می‌شوند، بهینه‌سازی‌های خاصی دارد که عملکرد برنامه‌ها را در سخت‌افزار Intel بهبود می‌بخشد.
- **دقت بالای بهینه‌سازی:** این کامپایلر به‌طور خاص در بهینه‌سازی کدهای محاسباتی و علمی که نیاز به عملکرد بالایی دارند، شناخته شده است.

معایب:

- **غیررایگان:** برخلاف GCC و Clang، ICC یک کامپایلر تجاری است و برای استفاده از برخی ویژگی‌های پیشرفته‌تر، باید هزینه پرداخت کنید.

۱-۹- مقایسه مزایای کامپایلرهای C++

عکس از حدول

فصل ۲

نحو و معناشناسی

۲ - ۱ - کلمات کلیدی

در ادامه فهرستی از ۴۸ کلمه کلیدی در زبان C++، توضیح مختصر و کاربرد آن‌ها همراه با مثال ارائه می‌شود:

۱. **int**

- **توضیح:** نوع داده عدد صحیح.
- **کاربرد:** تعریف متغیرهایی که اعداد صحیح را ذخیره می‌کنند.

۲. **float**

- **توضیح:** نوع داده اعشاری با دقت کم.
- **کاربرد:** ذخیره اعداد اعشاری کوچک.

۳. **double**

- **توضیح:** نوع داده اعشاری با دقت بالا.
- **کاربرد:** ذخیره اعداد اعشاری بزرگ‌تر.

۴. **char**

- **توضیح:** نوع داده کاراکتر.
- **کاربرد:** ذخیره یک کاراکتر.

۵. **bool**

- **توضیح:** نوع داده بولین .(true/false)
- **کاربرد:** ذخیره مقادیر منطقی.

۶. **void**

- **توضیح:** مشخص‌کننده بازگشت نداشتن توابع.
- **کاربرد:** تعریف توابعی که مقداری برنمی‌گردانند.

۷. **if**

- **توضیح:** شرطی.
- **کاربرد:** اجرای دستورات در صورت برقرار بودن شرط.

۸. **else**

- **توضیح:** شرط جایگزین.
- **کاربرد:** اجرای دستورات در صورت برقرار نبودن شرط.

۹. **switch**

- **توضیح:** انتخاب چندگانه.
- **کاربرد:** بررسی مقادیر مختلف یک متغیر.

۱۰. **for**

- **توضیح:** حلقه.
- **کاربرد:** تکرار دستورات با تعداد مشخص.

۱۱. **while**

- **توضیح:** حلقه.
- **کاربرد:** تکرار دستورات تا زمانی که شرط برقرار باشد.

۱۲. **do**

- **توضیح:** حلقه انجام بده سپس بررسی کن.
- **کاربرد:** حداقل یک بار اجرای دستورات.

۱۳. **return**

- **توضیح:** خروج از تابع و بازگرداندن مقدار.
- **کاربرد:** بازگرداندن مقدار در توابع.

۱۴. **break**

- **توضیح:** خروج از حلقه یا switch.
- **کاربرد:** خاتمه اجرای حلقه یا بلوک.

۱۵. **continue**

- **توضیح:** پرش به مرحله بعدی حلقه.
- **کاربرد:** ادامه اجرای حلقه با شرایط خاص.

۱۶. **class**

- **توضیح:** تعریف کلاس.
- **کاربرد:** تعریف اشیاء با خصوصیات و متدها.

۱۷. **public**

- **توضیح:** دسترسی عمومی.
- **کاربرد:** دسترسی آزاد به اعضای کلاس.

۱۸. **private**

- **توضیح:** دسترسی خصوصی.
- **کاربرد:** محدود کردن دسترسی به اعضای کلاس.

۱۹. **protected**

- **توضیح:** دسترسی محافظت‌شده.
- **کاربرد:** دسترسی محدود به کلاس و فرزندان آن.

۲۰. **struct**

- **توضیح:** تعریف ساختار.
- **کاربرد:** ایجاد گروهی از متغیرها.

۲۱. **const**

- **توضیح:** ثابت.
- **کاربرد:** تعریف مقادیری که تغییر نمی‌کنند.

۲۲. **namespace**

- **توضیح:** فضای نام.
- **کاربرد:** جلوگیری از تداخل نام‌ها.

```cpp
#include <iostream>

// Defining a namespace called "Math"
namespace Math {
    const double PI = 3.14159;
    double area(double radius) {
        return PI * radius * radius;
    }
}
// Defining another namespace called "Geometry"
namespace Geometry {
    const double PI = 3.14;  // Another value for PI, which could be used in geometry

    // Function to calculate the area of a square
    double area(double side) {
        return side * side;
    }
}

int main() {
    double radius = 5.0;
    double side = 4.0;

    // Using the area function in the Math namespace
    std::cout << "Area of circle: " << Math::area(radius) << std::endl;

    // Using the area function in the Geometry namespace
    std::cout << "Area of square: " << Geometry::area(side) << std::endl;

    return 0;
}
```

۲۳. using

- **توضیح:** استفاده از فضای نام.
- **کاربرد:** کاهش تایپ در استفاده از فضای نام.

۲۴. **try**

- **توضیح:** بلاک مدیریت خطا.
- **کاربرد:** آزمایش بخش کد حساس.

۲۵. **catch**

- **توضیح:** بلاک مدیریت خطا.
- **کاربرد:** گرفتن خطاها.

۲۶. **throw**

- **توضیح:** پرتاب خطا.
- **کاربرد:** تولید خطا در زمان اجرا.

۲۷. **enum**

- **توضیح:** نوع شمارشی.
- **کاربرد:** تعریف مقادیر ثابت مرتبط.

۲۸. **new**

- **توضیح:** تخصیص حافظه پویا.
- **کاربرد:** ایجاد شی یا آرایه در زمان اجرا.

۲۹. **delete**

- **توضیح:** آزادسازی حافظه پویا.
- **کاربرد:** جلوگیری از نشت حافظه.

۳۰. **this**

- **توضیح:** اشاره به شی فعلی.
- **کاربرد:** استفاده در متدهای عضو کلاس.

۳۱. **explicit**

- **توضیح:** جلوگیری از تبدیل ضمنی نوع.
- **کاربرد:** در سازنده‌ها برای جلوگیری از تبدیل‌های ناخواسته.

۳۲. **mutable**

- **توضیح:** اجازه تغییر به اعضای کلاس ثابت.
- **کاربرد:** برای اعضای داده‌ای که در متدهای const تغییر می‌کنند.

۳۳. volatile

- **توضیح:** نشان می‌دهد که متغیر ممکن است در هر لحظه تغییر کند.
- **کاربرد:** در برنامه‌نویسی سطح پایین و دسترسی به سخت‌افزار.

```cpp
#include <iostream>
#include <thread>
#include <atomic>

volatile bool stopFlag = false;

void threadFunction() {
    while (!stopFlag) {
        // Looping until stopFlag is true
    }
    std::cout << "Thread stopped.\n";
}

int main() {
    std::thread t(threadFunction);
    std::this_thread::sleep_for(std::chrono::
       seconds(1));
    stopFlag = true;
    t.join();
    return 0;
}
```

۳۴. inline

- **توضیح:** پیشنهاد اجرای توابع درون خطی به کامپایلر.
- **کاربرد:** برای بهبود کارایی در توابع کوچک.

```cpp
#include <iostream>

inline int add(int a, int b) { //
   inline
    return a + b;
}
int main() {
    std::cout << "Sum: " << add(3, 4) << '\n';
    return 0;
}
```

۳۵. register

- **توضیح:** پیشنهاد به کامپایلر برای ذخیره متغیر در رجیستر CPU.
- **کاربرد:** به ندرت استفاده می‌شود؛ عمدتاً تاریخی است.

۳۶. friend

- **توضیح:** اجازه دسترسی به اعضای خصوصی یا محافظت‌شده کلاس.
- **کاربرد:** تعریف توابع یا کلاس‌های دوست.

```cpp
#include <iostream>

class MyClass {
    private:
    int secretValue = 42;

    friend void revealSecret(const MyClass& obj); //

{;

void revealSecret(const MyClass& obj) {
    std::cout << "Secret value: " << obj.secretValue <<
        '\n';
}

int main() {
    MyClass obj;
    revealSecret(obj);
    return 0;
}
```

۳۷. constexpr

- **توضیح:** تعریف مقادیری که باید در زمان کامپایل ارزیابی شوند.
- **کاربرد:** برای بهینه‌سازی زمان کامپایل.

```cpp
    #include <iostream>

    constexpr int square(int x) { //
        return x * x;
    }
```

```
int main() {
    constexpr int value = square(5); //
    std::cout << "Square: " << value << '\n';
    return 0;
}
```

۳۸. **decltype**

- **توضیح:** تعیین نوع بازگشتی یک عبارت.
- **کاربرد:** معمولاً در متدهای قالبی استفاده می‌شود.

۳۹. **typename**

- **توضیح:** تعریف یا استفاده از نوع در کلاس‌های قالبی.
- **کاربرد:** برای اشاره به یک نوع در قالب‌ها.

۴۰. **static_cast**

- **توضیح:** تبدیل ایمن نوع در زمان کامپایل.
- **کاربرد:** جایگزین تبدیل‌های قدیمی C.

۴۱. **dynamic_cast**

- **توضیح:** تبدیل ایمن نوع در زمان اجرا.
- **کاربرد:** در کلاس‌های چندریختی استفاده می‌شود.

۴۲. **reinterpret_cast**

- **توضیح:** تبدیل نوع بدون تغییر بایت‌های داده.
- **کاربرد:** در تبدیل‌های سطح پایین.

۴۳. **static**

- **توضیح:** تعریف اعضای کلاس یا متغیرهایی که دامنه‌شان محدود است.
- **کاربرد:** ذخیره متغیرهایی که مقدارشان در تمام نمونه‌ها مشترک است.

۴۴. **typeid**

- **توضیح:** گرفتن اطلاعات نوع در زمان اجرا.
- **کاربرد:** برای بررسی نوع شیء.

```cpp
#include <iostream>
#include <typeinfo>  // Header for using typeid

class Base {
    public:
    virtual ~Base() {}  // Virtual function required for using typeid
};

class Derived : public Base {
};

int main() {
    Base* basePtr = new Derived();  // Create an object of type Derived and reference it with a Base pointer

    // Using typeid to get the type of the object at runtime
    std::cout << "Type of basePtr: " << typeid(*basePtr).name() << std::endl;

    // Without using a virtual pointer, the result will be the type Base
    std::cout << "Type of basePtr (without virtual): " << typeid(basePtr).name() << std::endl;

    delete basePtr;  // Freeing the allocated memory
    return 0;
}
```

۴۵. default

- **توضیح:** مقدار پیش‌فرض برای سازنده یا متد.
- **کاربرد:** استفاده در کلاس‌ها برای ساده‌سازی.

۴۶. override

- **توضیح:** مشخص می‌کند که متد بازنویسی‌شده است.
- **کاربرد:** در برنامه‌نویسی شیءگرا.

۴۷. final

- **توضیح:** جلوگیری از بازنویسی کلاس یا متد.
- **کاربرد:** امنیت در طراحی کلاس‌ها.

۴۸. alignas

- **توضیح:** مشخص کردن تراز حافظه.
- **کاربرد:** تنظیم حافظه برای بهینه‌سازی.

```cpp
#include <iostream>
#include <alignas>

struct alignas(16) MyStruct {
    int a;
    double b;
};

int main() {
    MyStruct s;
    std::cout << "Address of s: " << &s << std::endl;
    std::cout << "Alignment of MyStruct: " << alignof(
       MyStruct) << std::endl;
    return 0;
}
```