CPP

强制类型转换运算符，C的强转对应哪个?

c++里面的的类型转换符有四个，分别是`const_cast`、`static_cast`、`dynamic_cast`、`reinterpret_cast`。这四种类型分别有不同的作用。

- `const_cast`：一般用于去除const属性，将const转换为非const，但原变量的属性还是const，只是在强转的过程中会临时生成一份去除const属性的变量出来。
- `static_cast`：用于强制转换数据类型，但是没有运行时的安全检测，而且还不能进行交叉转换，意思就是不是同意继承体系的无法转换，常用于基本数据类型的转换吗、非const转换const
- `dynamic_cast`：使用最多的一个类型转换符，一般用于多态类型的转换，有运行时安全检测
- `reinterpret_cast`：属于比较底层的进制转换，没有任何类型检查和格式转换，仅仅是简单的二进制数据拷贝；可以交叉转换，可以将指针和整数相互转换。

而c语言的强转对应的就是static_cast，原因是他们两个在强转的时候都**不会进行类型强转的安全检测，类型不匹配的话就会报异常。**

智能指针

智能指针主要用于管理在堆上分配的内存，它将普通的指针封装为一个栈对象。当栈对象的生存周期结束后，会在析构函数中释放掉申请的内存，从而防止内存泄漏。C++ 11中最常用的智能指针类型为shared_ptr,它采用引用计数的方法，记录当前内存资源被多少个智能指针引用。该引用计数的内存在堆上分配。当新增一个时引用计数加1，当过期时引用计数减一。只有引用计数为0时，智能指针才会自动释放引用的内存资源。对shared_ptr进行初始化时不能将一个普通指针直接赋值给智能指针，因为一个是指针，一个是类。可以通过make_shared函数或者通过构造函数传入普通指针。并可以通过get函数获得普通指针。

在使用传统的指针的时候很容易出现以下问题：

- 需要手动管理内存
- 容易发生内存泄漏(忘记释放、出现异常等)
- 释放之后产生野指针

而只能指针的出现就是为了解决传统指针出现的问题，智能指针分别有：`shared_ptr`、`weak_ptr`、`unique_ptr`。

shared_ptr的原理

- 一个`shared_ptr`会对一个对象产生强引用
- 每个对象都有个与之对应的强引用计数，记录着当前对象被多少个

  ```
  shared_ptr
  ```

  强引用着
  - 可以通过`shared_ptr`的use_count函数会的强引用计数
- 当有一个新的`shared_ptr`指向对象时，对象的强引用计数就会+1
- 当有一个`shared_ptr`销毁时(比如说作用域结束)，对象的强引用计数就会-1

- 当一个对象的强引用计数为0时(没有任何`shared_ptr`指向对象时)，对象就会自动销毁(调用析构函数)。

使用`shared_ptr`时避免循环引用，循环引用会导致堆内存无法正确释放，导致内存泄漏。

weak_ptr的原理

`weak_ptr` 的实现原理同`shared_ptr`会差不多，不过`weak_ptr`产生的是弱引用，它可以用来解决`shared_ptr`的循环引用问题

unique_ptr的原理

- `unique_ptr`也会对一个对象产生强引用，它可以**确保同一时间只有1个指针指向对象**
- 当`unique_ptr`销毁时(作用域结束)，其指向的对象也就自动销毁了。
- 可以使用std::move函数转移`unique_ptr`的所有权(转移强引用对象)

指针和引用(区别)

- 引用必须被初始化，指针可以不初始化
- 引用不能改变被引用的对象，指针可以改变指向
- 引用没有自己独立的内存空间(指向变量内存)，指针有自己独立的内存
- 引用不能为空，指针可以为空

什么时候用引用，什么时候用指针

对于使用传递的值而不做修改的函数

- 数据对象较小，如内置数据类型或小型结构体，则按值传递

```
void func(int );
```

- 数据对象是数组，则只能使用指针，并将指针const指针

```
void func(const int *,int);//第二个参数为数组长度
```

- 数据对象是较大的结构体，则const指针或const引用都行

```
struct struc{…};
void func(const struc *);
void func(const struc &);
```

- 数据对象是类，则使用const引用

```
void func(const string &,);
```

对于使用传递的值要做修改的函数

- 数据对象是内置数据类型，则使用指针

```
void func(int *);
```

- 数据对象是数组，则只能使用指针

```
void func(int *,int);//第二个参数为数组长度
```

- 数据对象是结构体，则使用引用或指针

```
struct struc{…};
void func(struc *);
void func(struc &);
```

- 数据对象是类，则使用引用

```
void func(ostream &);
```

指针常量和常量指针

指针常量：const修饰指针本身

常量指针(只读类型的指针)：const修饰指针指向的类型

判断：const在*前还是后，前：常量指针；后：指针常量

指针常量：

- 必须初始化
- 可以通过指针修改指向变量的值
- 不可以改变指针的指向

常量指针：

- 可以不初始化
- 不可以通过指针修改指向变量的值
- 可以改变指针的指向
- 可以指向非const修饰的变量

左值引用和右值引用

首先要清楚左值和右值的概念。左值和右值的区分标准在于能够获取地址，能够获取地址便是左值，不能获取地址便是右值。

左值引用

传统的c++引用便是左值引用

```
int i = 1;
int &ii = i;
```

右值引用

c++11中增加了右值引用，右值引用关联到右值时，右值被存储到特定位置，右值引用指向该特定位置，也就是说，右值虽然无法获取地址，但是右值引用是可以获取地址的，该地址表示临时对象的存储位置。

```
int &&iii = 10;
```

右值引用可以解决两个问题：

- 临时对象非必要的昂贵的拷贝操作
- 在模板函数中如何按照参数的实际类型进行转发。

结构体struct和共同体union（联合）的区别

结构体：将不同类型的数据组合成一个整体，是自定义类型

共同体：不同类型的几个变量共同占用一段内存

结构体中的每个成员都有自己独立的地址，它们是同时存在的；

共同体中的所有成员占用同一段内存，它们不能同时存在；

sizeof(struct)是内存对齐后所有成员长度的总和，sizeof(union)是内存对齐后最长数据成员的长度、

结构体为什么要内存对齐呢?

平台原因（移植原因）：不是所有的硬件平台都能访问任意地址上的任意数据，某些硬件平台只能在某些地址处取某些特定类型的数据，否则抛出硬件异常

硬件原因：经过内存对齐之后，CPU的内存访问速度大大提升。

#define和const的区别

#define定义的常量没有类型，所给出的是一个立即数；const定义的常量有类型名字，存放在静态区域

处理阶段不同，#define定义的宏变量在预处理时进行替换，可能有多个拷贝，const所定义的变量在编译时确定其值，只有一个拷贝。

#define定义的常量是不可以用指针去指向，const定义的常量可以用指针去指向该常量的地址

#define可以定义简单的函数，const不可以定义函数

CAD

项目一

电子报批 2023.01-2024.01 (1年)

描述 园区电子报批系统，包含cad端全套操作，包括地块红线各类图层自动绘制，mfc数据输入界面记录，插件自动统计审核

职责 对图形数据记录查询修改，mfc界面全套制作，审核统计数据并渲染反馈

业绩 项目需求已全部完成，后续提出修改延伸

问题 1: 在CAD端自动生成地块红线图层的过程中，如何处理不同图层的数据输入和输出?

参考: 在 CAD 端自动生成地块红线图层时，需要处理不同图层的数据输入和输出。通过使用 AutoCAD 的 API，我们可以创建和管理多个图层。在输入数据时，将不同类型的数据分配到不同的图层上。在输出时，可以利用 AutoCAD 的命令和函数，将特定图层的数据导出为标准格式的文件，如 DXF 或 DWG 格式。具体实现包括调用 `AcDbLayerTableRecord` 类来创建图层，并使用 `AcDbLayerTable` 类进行管理。

问题 2: MFC界面全套制作过程中，如何实现用户数据输入的校验和异常处理?

参考: 在 MFC 界面开发过程中，为了实现用户数据输入的校验和异常处理，可以利用 MFC 的消息映射机制（Message Mapping）来捕获用户输入事件。通过编写事件处理函数，对输入数据进行校验。例如，可以在 `EN_CHANGE` 事件中检测用户输入的合法性，对于非法输入给予提示或阻止输入。此外，可以使用 `try-catch` 块来捕获异常，对异常情况进行适当的处理和记录。

问题 3: 你是如何实现MFC与CAD数据接口的？请描述具体步骤和使用的技术。

参考: 实现 MFC 与 CAD 数据接口的具体步骤如下：

1. **创建 COM 接口**：在 MFC 应用程序中，定义和实现一个 COM 接口，用于与 CAD 应用程序通信。
2. **注册 COM 组件**：将 MFC 应用程序注册为 COM 组件，使其能够被 CAD 调用。
3. **调用 AutoCAD API**：在 MFC 应用程序中，调用 AutoCAD 的 COM 接口，进行数据的读写操作。具体技术包括使用 `IAcadApplication` 和 `IAcadDocument` 接口来控制 AutoCAD，使用 `IAcadEntities` 接口操作图形实体。
4. **数据传输**：通过 COM 接口将用户输入的数据传输到 AutoCAD，在 AutoCAD 中进行相应的操作后，将结果返回给 MFC 应用程序。

问题 4: 在进行统计计算时，你如何优化SQL查询以提高性能?

参考: 优化 SQL 查询的步骤包括：

1. **创建索引**：为查询中频繁使用的列创建索引，以加快数据检索速度。
2. **优化查询语句**：重写查询语句，避免使用低效的操作。例如，尽量减少使用子查询，替换为连接（JOIN）。
3. **分区表**：对于大表，使用表分区（Partitioning）将数据分段存储，提高查询效率。
4. **缓存**：使用缓存机制，将常用的查询结果缓存起来，减少对数据库的访问次数。
5. **分析执行计划**：通过数据库的查询分析工具（如 SQL Server 的执行计划）查看查询的执行步骤，找出性能瓶颈，进行针对性的优化。

问题 5: 你如何利用多线程技术提高数据处理的并行度？请描述具体实现方法。

参考: 利用多线程技术提高数据处理的并行度，具体实现方法如下：

1. **创建线程池**：在 MFC 应用程序中，使用 C++ 标准库中的线程池或 Windows 提供的线程池 API，创建多个线程。
2. **任务分割**：将数据处理任务分割成多个独立的小任务，每个小任务可以在不同的线程中并行执行。
3. **同步与互斥**：使用同步机制（如互斥锁、信号量）保护共享数据，避免数据竞争和死锁。例如，可以使用 `std::mutex` 或 `CRITICAL_SECTION` 来实现线程同步。
4. **任务调度**：通过任务队列（Task Queue）和调度算法，将任务分配给空闲线程，提高处理效率。
5. **结果汇总**：所有线程完成各自任务后，将结果汇总并返回给主线程进行最终处理。

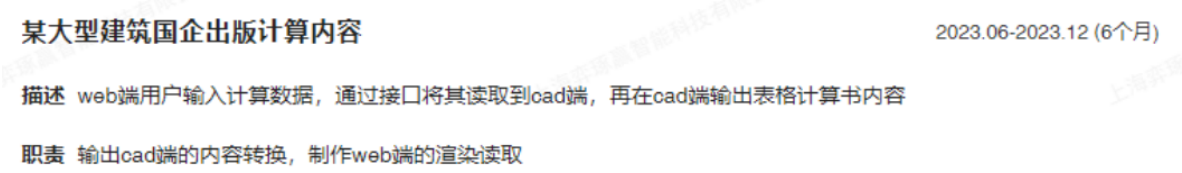
问题 6: 你如何在MFC中实现图形数据的高效渲染?

参考: 在 MFC 中实现图形数据的高效渲染，可以采取以下方法：

1. **双缓冲**：使用双缓冲技术，先将图形绘制到内存中的缓冲区，然后一次性输出到屏幕上，避免屏幕闪烁。
2. **GDI+**：使用 GDI+ 进行图形绘制，GDI+ 提供了更高效的图形处理功能和更多的绘制选项。

3. **减少重绘**：尽量减少不必要的重绘操作，只在需要更新的区域进行重绘，可以使用 `InvalidateRect` 函数指定需要重绘的区域。
4. **优化绘图算法**：对绘图算法进行优化，避免复杂度高的操作。例如，使用空间分割算法（如四叉树、八叉树）加快图形对象的查找和渲染。
5. **批量绘制**：将多个绘制操作合并为一个批量绘制操作，减少函数调用和上下文切换的开销。

项目二



问题 1: 在该项目中，如何实现从Web端获取用户输入数据并传输到CAD端?

参考: 在该项目中，从Web端获取用户输入数据并传输到CAD端，可以采用RESTful API或WebSocket技术实现。首先，在Web端使用前端框架（如React或Vue.js）创建用户输入界面，用户输入的数据通过AJAX请求或WebSocket发送到后端服务器。后端服务器可以使用Node.js或Python搭建，接收数据后，通过接口将数据传输到CAD端。CAD端可以使用AutoCAD的.NET API或ObjectARX开发接口读取数据并进行相应处理。

问题 2: 请描述CAD端如何处理从Web端接收的数据，并生成表格计算书内容?

参考: CAD端接收到从Web端传输的数据后，通过解析数据生成相应的图形和表格。具体步骤包括：

1. **数据解析**：将接收到的JSON或XML格式的数据解析成CAD能理解的格式。
2. **图形生成**：利用AutoCAD的API，根据解析后的数据生成对应的建筑图纸。
3. **表格生成**：根据数据生成表格内容，使用AutoCAD的Table对象创建表格，并填充数据。
4. **保存和输出**：将生成的图纸和表格保存为DWG文件，并提供给用户下载或进一步处理。

问题 3: 如何在Web端实现对CAD内容的渲染和读取?

参考: 在Web端实现对CAD内容的渲染和读取，可以使用WebGL或Three.js等图形库。具体步骤包括：

1. **CAD文件转换**：将CAD文件转换为WebGL或Three.js能够处理的格式，如OBJ或STL文件。
2. **文件传输**：将转换后的文件上传到服务器，并在Web端通过AJAX请求获取文件数据。
3. **渲染显示**：使用WebGL或Three.js库在Web页面上渲染CAD内容，提供旋转、缩放、平移等交互功能。
4. **数据读取**：通过JavaScript与后台通信，读取CAD文件中的数据，并在页面上显示。

问题 4: 在该项目中，遇到的最大技术难题是什么？你是如何解决的?

参考: 该项目中遇到的最大技术难题可能是如何高效地在Web端渲染复杂的CAD内容。解决方法包括：

1. **优化数据结构**：将CAD文件的数据结构优化为适合Web渲染的格式，减少数据量。
2. **增量加载**：使用增量加载技术，根据用户视图的变化，逐步加载和渲染CAD内容，避免一次性加载全部数据导致的性能问题。
3. **使用GPU加速**：利用WebGL或Three.js的GPU加速功能，提高渲染性能。
4. **分层渲染**：将CAD内容分层处理，只渲染用户当前需要查看的部分，减少渲染负担。

问题 5: 在项目中，你是如何确保数据传输的安全性和完整性的?

参考: 确保数据传输的安全性和完整性可以采取以下措施：

1. **使用HTTPS**：在Web端和后端服务器之间使用HTTPS协议，确保数据传输的加密性。
2. **数据验证**：在数据传输前后，对数据进行验证，确保数据的完整性和正确性。
3. **身份验证**：使用OAuth2或JWT等身份验证机制，确保只有授权用户才能访问和传输数据。
4. **数据备份**：在数据传输过程中，进行数据备份，防止数据丢失。
5. **日志记录**：记录数据传输的日志，方便问题排查和数据恢复。

项目三

mjc参数化绘图 2022.12-2023.07 (7个月)

描述 客户为日企微电子芯片行业，需求为输入参数直接出探针断面图等内容

职责 该项目中独立完成断面图全部内容，按照客户需求学习探针叠放方式并绘制，后续根据客户要求灵活修改输入数据等

业绩 完成项目并验收，客户效率有显著提升

问题 1: 在该项目中，如何实现参数输入直接生成断面图的功能?

参考: 为实现参数输入直接生成断面图的功能，首先需要设计一个用户友好的参数输入界面。利用 MFC 或 Qt 等 GUI 库创建界面，用户可以在界面上输入参数。接下来，通过算法将输入的参数转换为几何信息，利用绘图引擎（如 GDI+ 或 OpenGL）生成断面图。核心步骤包括：参数解析、几何计算、图形绘制。参数解析是指将用户输入的文本或数值解析为具体的绘图指令；几何计算是根据这些指令生成断面图的坐标和形状；图形绘制则是利用绘图库将这些几何信息绘制成图像。

问题 2: 你是如何根据客户需求设计探针叠放方式并绘制断面图的?

参考: 根据客户需求设计探针叠放方式需要详细理解客户的业务逻辑和具体需求。首先，通过与客户沟通，获取探针叠放的具体要求和标准。然后，将这些要求转化为数学模型或算法。在绘图过程中，根据这些模型或算法生成探针的位置和形状信息，并通过编程将这些信息绘制到断面图上。具体实现中，可能涉及使用递归或迭代算法来排列探针，并利用向量运算来确定探针的位置和方向。

问题 3: 在实现绘图功能时，遇到的最大挑战是什么？你是如何解决的?

参考: 在实现绘图功能时，遇到的最大挑战可能是如何高效地处理大量数据并确保绘图的准确性和性能。解决方法包括：

1. **数据优化**：对输入的数据进行预处理，去除冗余信息，确保数据简洁高效。
2. **算法优化**：使用高效的算法，如快速排序、空间分割等，来提高数据处理和绘图的速度。
3. **多线程**：如果绘图过程较为耗时，可以考虑使用多线程技术，将绘图任务分配到不同的线程中并行执行，提高整体效率。
4. **缓存机制**：对于重复的绘图操作，可以使用缓存机制，将已经绘制好的部分保存起来，以减少重复计算和绘图的时间。

问题 4: 你是如何灵活修改和输入数据的？请描述实现细节。

参考: 灵活修改和输入数据的实现细节如下：

1. **用户界面**：在用户界面设计上，提供友好的交互控件，如文本框、下拉菜单、滑动条等，方便用户输入和修改数据。

2. **数据绑定**：将用户输入的参数与后台数据模型绑定，确保用户修改参数时，数据模型能够实时更新。
3. **即时反馈**：在用户修改数据后，立即触发绘图更新，使用户能够看到修改后的效果。实现方法是通过事件驱动机制，当用户修改参数时，触发绘图函数重新计算并绘制图形。
4. **数据校验**：在数据输入时，进行实时校验，确保输入的数据合法有效。例如，可以在输入框中设置输入限制或正则表达式进行格式校验，防止用户输入非法数据。

问题 5: 该项目中使用了哪些技术栈？你是如何选择这些技术的?

参考: 该项目中使用的技术栈可能包括：

1. **编程语言**：C++ 作为主要的开发语言，具有高性能和强大的库支持。
2. **GUI 框架**：MFC 或 Qt 用于开发用户界面，MFC 集成在 Visual Studio 中，适合 Windows 平台开发；Qt 跨平台且功能强大，适用于多平台应用。
3. **绘图引擎**：GDI+ 或 OpenGL 用于绘图，GDI+ 适用于简单的 2D 图形绘制，OpenGL 则适用于复杂的 2D 和 3D 图形绘制。
4. **数据库**：SQLite 或 MySQL 用于存储和管理参数数据，SQLite 适合轻量级的本地数据库应用，MySQL 适合需要网络访问的大型数据库应用。
5. **算法库**：使用 Boost 或 Eigen 等库进行复杂的数学运算和算法实现。

选择这些技术主要基于项目需求、开发团队的技术背景和项目的性能要求。例如，C++ 和 MFC 组合适用于 Windows 平台高性能应用开发，Qt 则适用于需要跨平台支持的项目，OpenGL 适合需要高效绘图的项目。