地平线一面-规控岗-20230824

A.业务：项目+科研

B.技术基础

- **A*算法原理**

  答： A*算法是一种启发式搜索算法，结合了广度优先搜索和启发式函数来确定下一步移动的方向。它通过综合考虑实际路径代价和估计路径代价，选择具有最小代价值的节点作为下一个移动目标，从而找到最短路径。*A*中定义两个值：

  1. G值：表示从起始节点到当前节点的实际路径代价，也就是已经走过的路径长度。
  2. H值：表示从当前节点到目标节点的估计路径代价，也就是剩余路径的预估长度。 通过综合考虑起始节点到当前节点的实际路径代价G和当前节点到目标节点的估计路径代价H来选择下一个节点，并计算出F值：F = G + H，以确定下一步移动的方向。选择具有最小F值的节点作为下一步的移动目标。 在每次选择下一个节点时，A*算法会尽可能选择那些F值较小的节点，这使得算法能够尽可能快地找到最短路径，并避免对不必要的节点进行搜索。 当*A*算法达到目标节点或者无法找到可行解时，搜索过程结束。

- A*中的曼哈顿距离 $$d = |x1 - x2| + |y1 - y2|$$

C. c++基础

- **C++编译过程**

  答： C++的编译过程：预处理、编译、汇编、链接

  1. 预处理：编译器根据预处理指令（以#开头）处理源代码文件。预处理指令包括宏定义、条件编译等，预处理器会替换宏、展开头文件以及进行条件编译，生成一个经过预处理的源代码文件。
  2. 编译：将经过预处理的源代码文件转化成汇编代码。编译器会进行词法分析、语法分析和语义分析，将C++源代码转化成中间表示形式（如抽象语法树），然后生成对应的汇编代码。
  3. 汇编：将汇编代码转化成机器代码。汇编器会将汇编代码转化成机器指令的二进制表示形式，并生成目标文件（以.o为后缀）。
  4. 链接：将目标文件与其他所需的目标文件、库文件进行链接，生成可执行文件。链接器会进行符号解析、重定位等操作，将各个目标文件中的符号地址进行合并，生成最终的可执行文件。

- **C++ 内存泄漏**

  答： 内存泄漏是指程序在运行中动态分配的内存没有被完全释放，导致该内存无法再被程序访问和使用，从而造成内存资源的浪费。

  造成内存泄漏的主要原因有： 1.动态内存分配后未释放：使用new运算符分配内存，但忘记使用delete运算符释放内存。 2.释放了错误的内存：如果使用delete运算符释放了一个不是通过new运算符分配的内存，或者多次释放同一个内存块，都会导致内存泄漏或者程序崩溃。 3.指针赋值导致覆盖原有指针值：如果在动态分配内存后将新的地址赋给指针变量而忘记释放之前分配的内存，就会导致内存泄漏。 4.循

环引用导致内存无法释放：如果存在两个或多个对象之间互相引用，但没有正确断开引用关系，就会导致对象无法释放，在程序运行过程中造成内存泄漏。

- **构造函数可否是虚函数，为什么**

  答： 构造函数不能是虚函数。

  原因：

  1. 虚函数表：虚函数的调用是通过虚函数表（或称vtable）来实现的，每个类有自己的虚函数表，该表存储了虚函数的地址。对象的创建和销毁需要使用构造函数和析构函数，而此时对象还没有创建完成，因此还没有虚函数表。
  2. 虚函数调用机制：虚函数调用是基于对象的成员函数的地址来实现的，对象的地址由构造函数来初始化，如果构造函数是虚函数，那么在对象创建时，还没有确定虚函数的地址，也无法通过虚函数表进行调用。
  3. 虚析构函数：C++支持让基类的析构函数成为虚函数，这样在删除一个指向派生类对象的基类指针时，会调用派生类的析构函数。然而，构造函数不能是虚函数，这是由C++语法规定的。

- **虚函数的原理**

  答： 虚函数的原理是通过虚函数表（或称vtable）来实现的。每个包含虚函数的类都会有一个对应的虚函数表，该表存储了虚函数的地址。

  当创建一个对象时，编译器会为该对象分配内存空间，并在对象中插入一个指向虚函数表的虚函数指针（vptr）。这个指针指向类的虚函数表的起始地址。

  在使用基类指针或引用调用虚函数时，实际调用的是根据对象类型在虚函数表中找到的对应虚函数的地址。通过vptr指针找到虚函数表，然后再找到对应的虚函数地址，并进行调用。

- **虚函数表什么时候有的**

  答： 编译过程

- **派生与继承中，虚函数是怎么继承的**

  在继承关系中，派生类会继承基类的虚函数。 当派生类重写（override）基类的虚函数时，派生类将拥有自己的实现。 派生类可以选择保留或者覆盖基类的虚函数。

D.手撕代码

1.多线程下的单例模式 单例模式:一种设计模式，用于限制一个类只能创建一个对象实例。它可以确保在整个程序中只存在一个该类的实例，并提供一个全局访问点。

单例模式的核心思想:将类的构造函数声明为私有的，这样外部的代码就无法直接通过构造函数创建对象实例。同时，类内部提供一个静态方法，这个方法负责创建或获取类的唯一实例，并确保只有一个实例存在。静态方法会记录实例是否已经创建，如果已经创建则直接返回该实例，否则创建一个新的实例并返回。

```cpp
class Singleton {
private:
    static Singleton* instance;
    static std::mutex mtx;
    Singleton() {} // 将构造函数设为私有，禁止外部实例化

public:
    static Singleton* getInstance() {
        if (instance == nullptr) { // 如果实例不存在，则创建一个新实例
            std::lock_guard<std::mutex> lock(mtx);// 多线程下的单例模式
            instance = new Singleton();
        }
        return instance;
    }
};
```

2.查找子串 去字符串1中查找有没有字符串2的子串，如果有的话，返回字符串1中该子串的首地址 方法1：双层for循环

```cpp
bool my_strStr(const char* str1, const char* str2)
{
    int st1Len = static_cast<int>(strlen(str1));
    int st2Len = static_cast<int>(strlen(str2));
    bool match = false;
    for(int i = 0; i <= st1Len-st2Len;++i)
    {
      match = true;
       for(int j=0;j<st2Len;++j)
       {
           if(str1[i+j] !=str2[j])
           {
               match = false;
               break;
           }

           if(match)
           {
               return  match;
           }
       }
    }
    return match;
}
```

方法2：kmp