**8月9日作业**

**口 简述堆区和栈区的区别**

1. 管理方式：栈区由编译器自动管理，遵循“先进后出”原则，函数调用时分配栈空间，函数返回时自动释放；堆区需要程序员手动申请，若忘记释放会导致内存泄露。
2. 空间大小：栈区空间较小（通常几MB），由系统预先分配固定大小；堆区空间较大（可达GB级别），动态分配，大小 不固定。
3. 存储内容：栈区主要存储局部变量、、函数参数、返回地址等；堆区主要存储动态分配的对象、数组等需要长期存在的数据。
4. 分配效率：栈区分配效率高，通过栈指针移动实现，无需复杂操作；堆区分配效率低，需要查找空闲内存块。
5. 内存地址增长方向：栈区地址通常从高到低增长；堆区地址通常从低到高增长。

**口 编程实现：将两个float类型的值59.622f与92.6327f存入一个长度为4的char数组中，再从该 数组中提取出两个float类型的值，使用你能想到的方法，缩小提取出的小数和原来数字之间 的差距。**

**口 编程实现：已知一个矩阵结构体声明为：**

**typedef struct MParam {**

**u int8\_t row;**

**u int8\_t column;**

**float \*\*data;**

**}\*Matrix;**

**围绕这个结构体，使用C语言分配动态内存的方法编写一个动态矩阵运算库，实现必要功能，从而 实现：从键盘输入矩阵的行数、列数以及矩阵的每个元素，输出该矩阵与其转置矩阵的乘积。（矩 阵库中必须包含释放废弃矩阵空间的函数）**。

** 编程：使用qsort( ) 函数，从键盘输入任意数目的float类型的数据，回车后输出它们从小到大 排序的结果**。

口 **简述出现野指针的可能原因和相应的避免方法。**

可能原因：

1. 指针未初始化
2. 指针指向的内存被释放后未置空
3. 指针操作越界
4. 函数返回局部变量的地址

避免方法：

1. 指针定义时初始化
2. 释放内存后及时置空
3. 避免指针越界操作
4. 不返回局部变量地址
5. 使用智能指针
6. 加强代码检查

口  **分析该程序运行后的输出结果。**

int main() {

int arr[5] = { 1, 2, 3, 4, 5};

int\* ptr1 = (int\*)(&arr + 0x1);

int\* ptr2 = (int\*)((int)arr + 0x1);

printf("%x,%x", ptr1[-1], \*ptr2);

return 0;

}

Printf(“%x,%x”,ptr1[-1],\*ptr22)以十六进制输出：

Ptr1[-1]的值为5.十六进制为5

Ptr2的值为0x200000

因此，程序输出为：5，200000

口 **我们可以用#define定义常量，为什么要用枚举？**

1. 类型安全：枚举常量属于特定枚举类型，编译器会进行类型检查，#define无类型信息，易引发隐式转换错误
2. 作用域可控：枚举常量局限于枚举类型作用域，避免全局命名冲突，#define可能导致命名污染
3. 语义清晰：枚举能将相关常量组织在一起，体现逻辑关联，#define是零散定义，缺乏关联性
4. 调试友好：枚举常量在调试时显示标识符，而#define仅显示数值，可读性差

口 （选做/了解）使用C语言实现一个内存池，写好注释