## 一、C#编程

1-1. 以下代码意图是什么？有没有什么问题？如果有问题，应该怎么写？

|  |
| --- |
| for (int i=0; i<list.Count; i++)  {  if (list[i] == 9)  list.RemoveAt(i);  } |

**答**：意图是删除列表list中所有等于9的元素。

有问题。问题是顺序删除时，i++导致下标错位。

两种改法：倒序删除；调用RemoveAt(i)之后i--。（代码略）

1-2. LinkedList是什么？它的优缺点是什么？举例说明它的用途。

**答**：LinkedList是C#的链表类。

链表适合添加删除元素，时间复杂度O(1)。但链表不支持下标访问，而且内存空间不连续。

用途：需要经常插入删除节点的场合，例如用在贪食蛇游戏中，或实现对象池（如何用链表实现对象池？）。

1-3. private, public, protected, internal 四种修饰符分别是什么含义？

**答**：private：私有的，仅有类内部的代码能够访问。

public：公开的，类的内部外部均可访问。

protected：受保护的，类内部代码，或派生类可以访问。

internal：内部的，在当前程序集（当前库）内可以随意访问，其它人引用此程序集（库）时不可访问

1-4. 反射是什么，举例说明它的用途。

（视频讲解，案例见“反射\_装箱”的案例代码）

1-5. ref关键字有什么用途？它和out关键字有什么区别？

**答**：ref用于修饰函数参数，参数默认为值传递，加上ref变成引用传递。

out与ref类似，也是把参数变成引用传递。但out的参数只能作为输出参数，不能输入（也就是不能在函数中读取out参数传入的值）。

1-6. 阅读代码回答问题：

|  |
| --- |
| void Print(string str)  {  Debug.Log(str);  }  Action a；  a += () => Print(“A”);  Action b = a;  b += () => Print(“B”); |

1. 请写出a()的打印结果。
2. 请写出b()的打印结果。

**答**：第1问：A

第2问：A

B

1-7. 类和结构有什么区别？它们和堆、栈有什么联系吗？请举例常用的引用类型和值类型。

**答**：class类，是引用类型的。struct结构，是值类型的。

引用类型的对象，一定是在堆上创建。

值类型的对象，当它是局部变量时在栈上创建。

（想一想：Student类的分数int score，是值类型，它一般在栈上还是堆上？ 当值类型的变量属于某个对象时，那它就和对象在一起。）

常用的引用类型：字符串，自己定义的各种class。

常用的值类型：int，float等等，还有Vector3、Color等。

1-8. 描述接口（Interface）的用途。接口和抽象类（Abstract class）有什么相似和区别？

**答**：抽象类Abstract class与通常的类区别不大，只是限制该类不能实例化对象，只能用作基类。抽象类可以具有抽象函数，派生类一定要重写抽象函数。

接口与类完全不同，不同的类A和B，如果实现了同样的接口IExample，那么IExample类型的变量就可以指向A或B类型的对象，调用接口定义的方法。

接口不要求类之间有继承关系，但仍然可以统一处理。接口是对继承关系很好的补充。

1-9. Stack，Array，List，Hashset，Dictionary，Queue，分别说明它们的特点（区别）和应用场景。

附加题：说明通用object容器与泛型容器的优劣。（知识点：装箱，拆箱）

**答**：Stack，栈，先进后出。例如函数调用栈。

Array，数组。

List，列表。最常用的容器，可以添加、删除、插入元素，用于各种场合。

HashSet，哈希表，无泛型的字典。一般可以用Dictionary代替。

Dictionary，字典。优点是可以快速根据Key查找对象，用于需要快速查找对象的场合。

Queue，队列，先进先出。用于消息队列，还有依次处理数据的场合。

（视频讲解，案例见“反射\_装箱”的案例代码）

1-10. 举例说明lambda表达式的用途。

**答**：函数定义均可用lambda表达式代替，好处是让代码更简洁。例如：

List排序，自定义排序顺序。（请同学补充代码）

DOTween动画，定义回调函数。

定义各种事件处理函数。

1-11. GC是什么意思？在代码设计中为什么要考虑它？

**答**：GC是垃圾回收（Garbage Collection）的意思。

C#等现代语言支持自动垃圾回收，当一个对象不再被任何变量引用时，就会被作为垃圾回收。

垃圾回收简化了编程，但是GC过程要消耗不小的CPU资源，影响程序执行效率，严重时导致游戏卡顿。所以要尽可能减少GC。

1-12. （上机题）要从字典中删除一个指定的元素，实现这个函数：

void RemoveItem(Dictionary<int, Item> dict, string itemName)

其中Item是道具类，道具具有名称Item.name。

**答**：新版本的C#支持在遍历字典时直接删除元素。

老版本的C#要把想删除的Key先记录在一个List里，然后再遍历List删除字典中的元素。

（答题时在简单说明后补任意一种写法的代码）

1-13. 某个类定义为：

|  |
| --- |
| class Student {  string name;  int score;  } |

有一个列表List<Student> students，请分别针对name和score进行排序。

**答**：（常见题目，正课讲过多次，请同学们补充代码。）

1-14. 说明值类型和引用类型赋值的区别。

**答**：（直接写代码说明最明白）

Vector3 a = new Vector3(1,2,3);

Vector3 b = a;

这时a和b是两个独立的Vector3对象，修改a的值不会改变b的内容。

而如果是引用类型，则只创建了一个对象，a和b变量都指向同一个对象。修改a对象，则b也会发生变化。

1-15. 位运算有哪些？举例说明它的用途。

**答**：按位与 &

按位或 |

按位异或 ^

按位取反 ~

按位左移和右移 << >>

（二进制操作常用于Unity中的Layermask计算；或者用一个int类型的变量保存多个bool信息；或者将两个较小的坐标值压缩在一个int中。任意举出一两个综合实例即可。）

1-16. 简述C#源文件编译过程

*C#源码 > CIL > 在虚拟机上运行*

*IL2CPP：C#源码 > CIL > C++源码 > 编译，以原生(Native)方式运行*

（需补充通识课程）

## 算法

2-1. 在你将要开发的手游中，要求体力每5分钟增加1点。我们可以通过int GetTimestamp()函数获得当前的时间戳，请问如何实现此需求。具体来说，要保存什么数据，用什么样的方法刷新体力？

**答**：（时间戳是指现在距离1970年1月1日的秒数。这里不能直接读取Unity的Time.time，是因为回复的逻辑在不进行游戏的时候依然要计算，需要用一种在不玩游戏时也能正确计算的算法。）

设int now = GetTimestamp()，记录上次回复时间 last = 0;

(now - last) / (5\*60) 就是需要回复体力的点数。

回复体力后，设置last = last + (回复点数 \* 300); 或者 last = now - (now - last) % (300)

（请同学画时间轴思考，验证答案）

2-2. 在上述问题的基础上，有一个限制参加活动次数的需求，玩家可以如何实现每天0点刷新活动次数？活动次数为一天2次。

如何实现每天5点刷新活动次数？

**答**：（提示：考虑now / (24\*3600) 是什么含义？

（记录上一次回复的总天数为last\_day，当天数不同时，就可以重置活动次数了。）

（0点改为5点，从平移坐标轴的角度考虑：如果把所有的时间都减去5小时，就能用上一问的方法解决。这一问想明白对提升思维帮助很大。）

2-3. （上机题）用深度优先搜索算法在Unity的场景树中查找名称为xxx的物体，该物体当前是隐藏状态。提示：得到所有根节点物体写法——

UnityEngine.SceneManagement.SceneManager.GetActiveScene().GetRootGameObjects()

|  |
| --- |
| GameObject Find(Transform trans, string name) // 递归查找name物体  {  if (trans.name == name) { return trans; }  foreach (var child in trans)  {  Transform temp = Find(child, name);  if (temp != null) { return temp; }  }  return null;  }  调用时：  GameObject[] objs =  UnityEngine.SceneManagement.SceneManager.GetActiveScene().GetRootGameObjects();  foreach (var obj in objs)  {  Transform ret = Find(obj.transform, “xxx”);  if (ret != null)  {  break;  }  }  ret即为要找的物体，如果ret==null表示没找到。 |

（用递归实现的深度优先搜索，请同学测试验证。）

2-4. 以下程序的输出是什么？

|  |
| --- |
| Class Program  {  Static void Main(string[] args)  {  Console.WriteLine(func(99));  }  Public static int func(int x)  {  int countx = 0;  while (x>0)  {  x = x & (x-1);  countx++;  }  return countx;  }  } |

**答**：可以计算出一个整数的二进制表示中，包含多少个1。

常规算法中，一个int必须遍历32个bit，才能算出有多少个1。

而这个算法中，0b1001仅循环2次，就能算出有2个1。循环次数仅与1的个数有关。

（请同学代入一个数字尝试，发现算法的原理。）

2-5. 用两个Stack实现一个Queue，将代码写在下面。

**答**：（老师画图说明，请同学补充代码）

2-6. 一个列表，其中只有一个元素出现了奇数次，其它元素都出现了偶数次。找出这个奇数次的元素并输出。

例如：[1,1,3,3,4,5,4]

**答**：常规做法是用字典或集合（Set）记录出现过的元素。

这里用二进制异或可以不用额外的空间直接得出结果。（而创建字典需要额外占用内存空间）

|  |
| --- |
| int s = 0;  foreach (int n in array)  {  s ^= n;  }  Console.WriteLine(s); |

2-7. 描述快速排序的流程。另外，它的时间复杂度是多少（可以给出范围）？它是稳定排序还是不稳定排序？

**答**：（先查阅资料写一个快速排序，理解递归过程。）

快速排序的时间复杂度介于O (n·log2n) 和 O(n2) 之间。

快慢取决于选取中间数时，中间数离中位数较近还是较远。

快速排序属于不稳定排序。（稳定的意思，是排序后同样的数字前后顺序是否发生变化）。

2-8. 使用快排时，何时会达到最快的情况，或最慢的情况？

**答**：快慢取决于选取中间数时，中间数离中位数较近还是较远。

（这里可以详细描述）

2-9. 有一个二叉树，其节点定义为：

|  |
| --- |
| class BinNode  {  int value;  BinNode left;  BinNode Right;  } |

给你某个二叉树的头节点BinNode head；遍历它的所有节点并打印value。

**答**：

|  |
| --- |
| // 深度优先搜索，遍历树  static void PrintTree(TreeNode node)  {  if (node == null) { return; }  Console.WriteLine(node.value);  PrintTree(node.left);  PrintTree(node.right);  return;  } |

（如果是深度优先，遍历二叉树的代码非常简单。但构造二叉树一定要写写看）

附：构造二叉树的代码：

|  |
| --- |
| static void AddNode(TreeNode node, int? left, int? right)  {  if (left != null)  {  node.left = new TreeNode((int)left, null, null);  }  if (right != null)  {  node.right = new TreeNode((int)right, null, null);  }  }  static void Main(string[] args)  {  // 构造一棵二叉树  TreeNode root = new TreeNode(1, null, null);  AddNode(root, 2, 3);  TreeNode left = root.left;  TreeNode right = root.right;  AddNode(left, 4, 5);  AddNode(right, 6, 7);  AddNode(left.left, 8, 9);  AddNode(left.right, 10, 11);  ……………… |

2-10. 有一个自定义的单链表节点，结构为：

|  |
| --- |
| class LinkedNode  {  int value;  LinkedNode next;  } |

给定你一个链表的头部：LinkedNode head。完成以下两个函数：

|  |
| --- |
| // 在node之前插入一个节点，值为v  static void AddNode(LinkedNode head, LinkedNode node, int v);  // 删除这个节点node  static void RemoveNode(LinkedNode head, LinkedNode node); |

（链表的课上有讲过）

2-11. 生成一个指定长度的随机字符串，要求它只包含大小写字母和数字。

**提示**：

|  |
| --- |
| string s = “abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890”; |

从s中随机取字符即可。

2-12. 你在实践中用过哪些数据结构？用在什么场景下？

**答**：List用过很多，比如保存技能列表，保存很多物体。

Dictionary，比如把Json数据读取到字典里。

LinkedList链表，用于贪食蛇游戏。

Queue，用于搜索算法。

2-13. 二叉树的高度为H，求最多节点数？二叉树的节点个数为N，求最小高度。

**提示**：二叉树就是每个节点最多只有两个子节点的树。

如果节点个数为N，怎样能让树最矮？

如果填满高度H的二叉树，能填多少个节点？

2-14. 描述字典的内部实现原理。

**提示**：正式课讲过。

**关键词**：哈希算法。字典内部是数组，元素为键值对，利用下标快速定位。

## Unity实践

3-1. 有一个继承自MonoBehaviour的类A，请写出它的单例模式。

|  |
| --- |
| class A : Monobehaviour  {  public Instance {get; private set;}  void Awake()  {  Instance = this;  }  } |

当挂载了A脚本的物体置于场景中时，A.Instance才会生效，否则Instance为空。

3-2. 举例说明对象池的 作用。如果让你设计一个对象池类，它应当有哪些主要字段和方法？

**提示**：参考书第13.1节。

必要的字段：一个存放所有对象的容器。

必要的方法：生成物体和销毁物体方法。

但实际上不会销毁物体，会隐藏起来方便下次创建再显示出来。

3-3. 举例说明状态机在游戏中的使用。行为树和状态机的区别和联系。

**答**：状态机用处很多，比如动画状态机、AI状态机都很常用。

当玩家操作有不同的阶段，或游戏分为不同阶段时，也可以使用状态机。

行为树用来实现更加复杂的AI系统。相比状态机，行为树没有明确的状态，可以实现基于优先级的、多分支的AI逻辑。

3-4. 举例Unity的四个特殊文件夹，它们分别是做什么用的？

**答**：1. Resources文件夹，其中的资源一定会被打包无论是否用到。可以动态加载其中的资源。

1. Plugins文件夹，存放程序插件，比如LitJson库、DOTween的dll都放在这里。Plugins支持区分不同的平台，比如x86、arm等等。
2. Editor文件夹。Editor文件夹可以不止一个，只要名字是Editor的文件夹，都会被Unity当作编辑器专用文件夹，里面放编辑器专用的资源和脚本，不会被打包到最终的游戏中。
3. StreamingAssets文件夹，里面的文件会原封不动出现在打包后的游戏目录中，用于存放一些不希望被Unity压缩的特殊资源。

3-5. Unity协程和线程的区别？

**提示**：务必阅读参考书13.2节。理解到Unity的协程是一种模拟同时执行，实际上每帧按顺序执行的手段。

而线程是操作系统的概念，线程是操作系统提供的，负责运行程序的最小单元。多个线程可以同时执行，如果多个线程同时读写了同一个数据，可能造成数据冲突。

3-6. 描述AssetBundle的用法。

3-7. 描述Unity中资源热更新的原理和流程。

**提示**：这两道题请同学阅读和实践参考书第9章，本章对于理解Unity的设计架构非常重要，9.2节还探讨了热更新的原理。

对于热更新，要求同学们从概念上和从AB包操作的角度理解它。在面试笔试时回答自己知道的部分。

3-8. 举一个例子，说明如何用射线检测来实现碰撞。

**提示**：开放式问题，题干不全，可以举一个例子来描述即可。

在你描述的基础上面试官可能会进一步问问题。

3-9. unity资源中每个文件、文件夹会对应一个meta文件，请问meta文件作用是什么？如果你和同事电脑中某个文件的meta文件不一致，会有什么影响？

**提示**：阅读本书第9.1节。

这个问题在实际开发中很重要，因为偶尔会遇到meta文件被修改的情况。

meta文件内有资源的编号ID，以及资源的导入配置。当meta文件出问题时，配置好的参数和引用会消失。

当一个图片的meta文件，两台电脑上不一样时，那么其它任何用到这个图片的地方，都会引用不同的ID。在合并工程时出现引用错误的情况。

3-10. Unity里面Vector3为什么要设计成值类型

**答**：在游戏中，我们经常会在函数中创建很多临时的Vector3变量用于向量计算。

如果Vector3是引用类型，就会带来大量零碎的GC。

而Vector3作为struct值类型，在栈上创建Vector3是不会在堆上分配内存的，对性能影响小。

另外，值类型在赋值、修改时也比较方便。

## 图形学基础与Unity优化

4-1. Unity中，sharedmaterial和material有什么区别。

meshRenderer.sharedmaterial = xxxx;

meshRenderer.material =xxx; // 有新建材质球的含义

4-2. mipmap是什么？它如何使用，有什么作用？

**答**：mipmap就是提前给贴图生成边长缩小2倍、4倍、8倍……的一系列贴图。这样当物体较远，需要的贴图较小时，自动匹配较小的贴图。

这样做一方面能降低贴图精度，改善性能；二是在渲染小物体时贴图更正确。

4-3. DrawCall是什么，如何降低DrawCall？

（*推荐图书：《Unity游戏优化（第2版）》*）

**提示**：CPU与GPU协作时，CPU将渲染相关的指令、数据传输给GPU，每提交并执行一次命令，就是一次DrawCall。

DrawCall给CPU带来额外的负担，如果一次DrawCall能多处理一些渲染，降低DrawCall次数，就能优化渲染性能。

降低DrawCall的手段：合批（Batch，包括动态合批和静态合批），合批就是把能够一起处理的操作统一处理。

复用材质球，减少场景中的半透明物体、半透明UI，都能降低DrawCall。

4-4. 三阶矩阵（3x3矩阵）乘以向量，能否实现三维向量位移操作？

**答**：不能，三阶矩阵只能实现线性变换。所谓线性变换就是原点不动的前提下，进行旋转、缩放、斜切等变换。线性变化不能改变原点，所以无法实现位移操作。

引擎中必须采用三维齐次矩阵（4x4矩阵），才能实现位移变换。

## 智力测试

5-1. 小王、小张、小赵三个人是好朋友，他们中间一个人下海经商，一个人考上重点大学，一个人参军了。此外他们还知道以下条件：小赵的年龄比士兵大；大学生年龄比小张小；小王的年龄和大学生年龄不一样。

请推出这三个人的身份。

**思考过程**：

小赵的年龄比士兵大 => 小赵不是士兵，只能是（经商，大学生）

大学生年龄比小张小 => 小张不是大学生，只能是（经商，士兵）

小王的年龄和大学生年龄不一样 => 小王不是大学生，只能是（经商，士兵）

年龄：小赵 > 士兵，小张 > 大学生

假设：小赵=大学生 => 大学生 > 士兵 => 小张≠士兵 => 小张 = 经商 => 小王=士兵

即：小赵=大学生，小张 = 经商，小王=士兵。

验证可行，不知道有没有其它可能的情况。

5-2. 有三十六头猪赛跑、但是赛道一次只能跑六只，问最少要跑几次能选出跑的最快的三只。

**提示：**

1. 分6组跑6次，能够淘汰哪些猪？

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| o | o | o | o | o | o |
| o | o | o | o | o | o |
| o | o | o | o | o | o |
| x | x | x | x | x | x |
| x | x | x | x | x | x |
| x | x | x | x | x | x |

1. 每组第一名（正好6只），一起跑一次，得到排名。能够淘汰哪些猪？

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| o | o | o | o | o | o |
| o | o | o | o | o | o |
| o | o | o | o | o | o |
| x | x | x | x | x | x |
| x | x | x | x | x | x |
| x | x | x | x | x | x |

只要想通第2步，答案就出来了。