数据结构期中大作业

姓名：杨瑞灵 学号：2252941

1.基本概念（20%）

（1）请给出你对数据结构、逻辑结构、物理结构、算法等概念的理解和认识。

- 数据结构:
  - ***定义***：**数据结构是以某种方式联系在一起的数据元素的集合。**程序中的数据结构反映了程序员**在程序中表示信息的方法。**

    结构 = 实体 + 关系

  - ***研究***：数据结构研究的是数据元素之间抽象化的相互关系及这种关系在计算机中的存储表示。对每种结构定义各自的运算，设计出相应的算法，并用某种语言实现该算法。
  - ***举例***：常见的数据结构包括（逻辑结构）
    - 集合
    - 线性结构：线性表、链表、栈、队列
    - 树形结构
    - 图状结构

    eg．二元组(D,S）其中D是数据元素的有限集，S是D上关系的有限集

- 逻辑结构:
  - ***定义***：**数据与数据之间的关系**，逻辑结构是从逻辑关系上描述数据。它关注的是数据之间的逻辑关系，而不关注数据的存储形式。
  - ***举例***：主要包括集合、线性结构、树结构、图结构等（数据结构）。
- 物理结构:
  - ***定义***：物理结构又叫存储结构，它是指 **数据结构在计算机中的存储表示。**
  - ***举例***：常见的物理结构包括
    - 顺序存储结构
    - 链式存储结构
    - 索引存储结构
    - 散列存储结构

同样是线性表（逻辑结构），它既可以线性存储，也可以链式存储（物理结构）。

选择什么样的物理结构取决于数据的特点

eg. 一元多项式：

如果次数与项数差不多，0系数较少，且需要存储的数据较少可以用线性结构，数组的下标表示次数，存储系数

如果0系数太多，用链式存储更节省空间

- 算法:
  - ***定义***：**处理问题的方法。**算法是解决特定问题求解步骤的描述，在计算机中为一个有限的指令集。
  - 它包括输入、输出、有穷性、确定性和可行性五个基本要素。

（2）请给出你对数据结构、逻辑结构、物理结构、算法等概念之间关系的理解和认识。

- **逻辑结构和物理结构**是数据结构的两个方面，它们共同构成数据结构

数据结构 = 数据 + 结构
结构 = 逻辑结构 (+ 物理结构)

- **算法**则在某种**数据结构**的基础上实现特定功能的步骤描述。算法的效率往往受到所选数据结构的影响。
- 反之，算法也决定了如何构造数据

程序 = 算法 + 数据结构
算法建立在数据结构上
算法决定构造数据的方法

- **任何一个算法的设计取决于选定的数据(逻辑)结构,而算法的实现依赖于采用的存储结构**
- 数据结构、逻辑结构、物理结构和算法之间有紧密的相互关系，它们共同构成了计算机科学中的基本框架，用于有效地处理和管理数据。

2. 请给出一个适合用线性表求解的应用问题。要求：（1）给出所采用的存储结构以及理由；（2）该应用问题有哪些基本操作，并简要说明算法思想。（40%）

应用问题：商品库存管理

存储结构：数组，因为数组能够通过索引快速访问任意元素，便于库存查询和更新。

设计思想：

将商品信息以及库存数量作为一个整体，定义为商品节点。

使用数组来存储所有的商品节点，每一个商品节点在数组中占据一个位置。

通过数组索引来快速定位和访问商品节点，以实现库存的快速查询和更新。

设计过程：

定义商品节点的数据结构。

创建一个数组来存储商品节点。

实现基本操作的函数，如添加、查询、更新和删除商品信息。

存储结构代码示例：

```
typedef struct {
    char name[50];
    int stock;
} Item;

Item inventory[100];
int count = 0;
```

基本操作代码示例：

- 添加商品信息

```
void addItem(char *name, int stock) {
  strcpy(inventory[count].name, name);
  inventory[count].stock = stock;
  count++;
}
```

- 查询商品信息

```
void queryItem(char *name) {
  for(int i = 0; i < count; i++) {
      if(strcmp(inventory[i].name, name) == 0) {
          printf("Item: %s, Stock: %d\n", name, inventory[i].stock);
          return;
      }
  }
  printf("Item not found\n");
}
```

- 更新商品库存

```
void updateStock(char *name, int newStock) {
  for(int i = 0; i < count; i++) {
      if(strcmp(inventory[i].name, name) == 0) {
          inventory[i].stock = newStock;
          printf("Stock updated successfully\n");
          return;
      }
  }
  printf("Item not found\n");
}
```

- 删除商品信息

```
void deleteItem(char *name) {
  int index = -1;
  for(int i = 0; i < count; i++) {
      if(strcmp(inventory[i].name, name) == 0) {
          index = i;
          break;
      }
  }
  if(index != -1) {
      for(int i = index; i < count - 1; i++) {
          inventory[i] = inventory[i + 1];
      }
      count--;
      printf("Item deleted successfully\n");
  } else {
      printf("Item not found\n");
  }
}
```

3.请给出适合用栈求解的一个应用问题。要求：

（1）给出所采用的存储结构以及理由；

（2）该应用问题有哪些基本操作，并简要说明算法思想。（40%）

应用问题：迷宫路径寻找

存储结构：循环队列，因为循环队列可以高效地利用空间，避免队列前端的空间浪费。

设计思想：

- 通过**宽度优先搜索**(BFS)来寻找从起点到终点的最短路径。
- 使用**队列**（deque）来保存待处理的位置，初始化时将起点加入队列。
- 在搜索过程中，每次从队列中取出一个位置，然后尝试向四个方向（上、下、左、右）移动，新的位置加入队列，并标记已访问，以防止重复处理。
- 直到找到终点或队列为空（无解）为止。

设计过程：

- 定义存储单元的数据结构，包含行和列的坐标。
- 定义循环队列的数据结构，包括队头、队尾指针和队列大小。
- 实现循环队列的基本操作，包括入队、出队和判断队列是否为空。
- 实现迷宫的宽度优先搜索算法，包括初始化、处理队列中的每个位置，直到找到终点或无解。

存储结构代码示例：

```
typedef struct Position {
int row;
int col;
} Position;

typedef struct {
Position queue[100];
int front;
int rear;
int size;
} Queue;

void initialize(Queue\* q) {
q->front = 0;
q->rear = 0;
q->size = 0;
}

int isEmpty(Queue* q) {
return q->size == 0;
}
```

基本操作代码示例:

- 入队

  ```
  void enqueue(Queue* q, Position pos) {
  if(q->size == 100) {
  printf("Queue overflow\n");
  exit(1);
  }
  q->queue[q->rear] = pos;
  q->rear = (q->rear + 1) % 100;
  q->size++;
  }
  ```

- 出队

```
Position dequeue(Queue\* q) {
if(isEmpty(q)) {
printf("Queue underflow\n");
exit(1);
}
Position pos = q->queue[q->front];
q->front = (q->front + 1) % 100;
q->size--;
return pos;
}
```

宽度优先搜索算法可以根据具体的迷宫布局和规则进行设计。通常包括读取迷宫布局、初始化队列结构、从起点开始，不断处理队列中的每个位置，直到找到终点或确定无解。在这个过程中，队列结构用于保存待处理的位置，以便在广度优先搜索过程中按照先入先出的原则处理每个位置。