8.7 作业

程序说明

- 从数据结构角度看，栈和队列是线性表，其特殊性在于栈和队列的基本操作是线性表操作的子集，它们是操作受限的线性表。

栈

> 栈（stack）是一种遵循先入后出逻辑的线性数据结构。
>
> - 栈（Stack）是限定只能在表尾进行插入和删除操作的线性表。
> - 在表中，允许插入和删除的一端称作“栈顶(top)”，
> - 表头端称作"栈底(bottom)" 。
>
> - 栈必须按“后进先出”的规则进行操作
> - 栈只允许在表尾一端进行插入和删除
> - 通常称往栈顶插入元素的操作为“入栈”
> - 称删除栈顶元素的操作为“出栈”
> - 因为后入栈的元素先于先入栈的元素出栈，故被称为是一种"后进先出"的结构，因此又称LIFO（Last In First Out）表。

队列

> 队列（queue）是一种遵循先入先出规则的线性数据结构。
>
> - 队列（queue）是一种先进先出的线性表，限定只能在表的一端进行插入和在另一端进行删除操作。
> - 在表中，允许插入的一端称做“队尾(rear)”
> - 允许删除的另一端称做“队头(front)”。
> - 队列又称FIFO（First In First Out 的缩写）表。
>
> - 假溢出现象
>
> 在顺序队列中，当尾指针已经到了数组的上界，不能再有入队操作，但数组中还有空位置，这就叫“假溢出”。
>
> - 解决假溢出的途径
>
> ——采用循环队列

代码实现

栈

栈常用的实现方式为:

1. 基于链表实现
2. 基于数组实现
   这里采用培训的方式进行实现

```
#include "inc/stack.h"

// 栈的初始化
void stack_init(stack_t *stack,stack_size_t size){
    stack->base = (char*)malloc(size);//分配内存，并将其地址赋给栈底
    stack->top = stack->base;//初始时栈顶等于栈底
    stack->size = size;//赋值栈的大小
}

bool stack_empty(stack_t *stack){
    return stack->top == stack->base;//栈顶等于栈底时，栈为空
}

bool stack_full(stack_t *stack){
    return (stack->base - stack->top) == stack->size;//栈底减去栈顶等于栈
的大小时，栈满
}

bool stack_push(stack_t *stack, void *value,stack_size_t size){
    if(stack_full(stack)){
        return false;
    }//栈满时，返回false，不进行入栈操作
    stack->top -= size;//栈顶减去size，即栈顶向下移动size个单位
    memcpy(stack->top,value,size);//将value的内容拷贝到栈顶
    return true;
}

bool stack_pop(stack_t *stack, void *value,stack_size_t size){
    if(stack_empty(stack)){
        return false;
    }//栈空时，返回false，不进行出栈操作
    memcpy(value,stack->top,size);//将栈顶的内容拷贝到value
    stack->top += size;//栈顶向上移动size个单位
    return true;
}

bool stack_top(stack_t *stack, void *value,stack_size_t size){
    if(stack_empty(stack)){
```

```
        return false;
    }//栈空时，返回false，不进行取栈顶操作
    memcpy(value,stack->top,size);//将栈顶的内容拷贝到value
    return true;
}

void stack_delete(stack_t *stack){
    free(stack->base - stack->size);//从头开始释放栈的连续内存
}

int main(){
    stack_t stack;
    stack_init(&stack,10);

    float a = 10.f;
    double d = 30.f;
    int b = 20;

    stack_push(&stack,&a,sizeof(float));
    stack_push(&stack,&d,sizeof(double));
    stack_push(&stack,&b,sizeof(int));

    float a2;
    double d2;
    int b2;

    stack_pop(&stack,&b2,sizeof(int));
    stack_pop(&stack,&d2,sizeof(double));
    stack_pop(&stack,&a2,sizeof(float));

    printf("%f %d %lf\n",a2,b2,d2);

    stack_delete(&stack);
    return 0;
}
```

队列

与栈类似，队列常用的实现方式为:

1. 基于链表实现
2. 基于数组实现
   这里采用培训的方式进行实现
   ps:注意，实现循环队列常规方法仍然需要多耗费一个空间去解决“假溢出”，培训时这个地方没有很好的解决

```
#include"inc/queue.h"

void queue_init(queue_t *queue){
    queue->r = queue->w = 0;//初始化队列的头尾指针
}

bool queue_enqueue(queue_t *queue,float value){
    if(queue_is_full(queue)){
        return false;
    }//队列满时，返回false，不进行入队操作
    queue->data[queue->w] = value;//将value赋值给队列的尾指针
    ++queue->w;//尾指针向后移动一个单位
    // queue->w %= QUEUE_MAX_SIZE;
    queue->w %= (QUEUE_MAX_SIZE+1);//取余，防止越界
    return true;
}

bool queue_dequeue(queue_t *queue,float *value){
    if(queue_is_empty(queue)){
        return false;
    }//队列空时，返回false，不进行出队操作
    *value = queue->data[queue->r];//将队列的头指针的值赋给value
    ++queue->r;//头指针向后移动一个单位
    // queue->r %= QUEUE_MAX_SIZE;
    queue->r %= (QUEUE_MAX_SIZE+1);//取余，防止越界
    return true;
}

bool queue_front(queue_t *queue,float *value){
    if(queue_is_empty(queue)){
        return false;
    }//队列空时，返回false，不进行取队头操作
    *value = queue->data[queue->r];//将队列的头指针的值赋给value
    return true;
}

bool queue_is_empty(queue_t *queue){
    return queue->r == queue->w;//头指针等于尾指针时，队列为空
```

```
}

bool queue_is_full(queue_t *queue){
    // int sz=queue->w-queue->r;
    // sz=sz>=0?sz:sz + QUEUE_MAX_SIZE;
    // return sz == QUEUE_MAX_SIZE;
    return (queue->w + 1) % (QUEUE_MAX_SIZE+1) == queue->r;//尾指针加1取
余等于头指针时，队列满
}

int main(){
    queue_t queue;
    queue_init(&queue);
    for(int i =0;i< QUEUE_MAX_SIZE;i++){
        queue_enqueue(&queue,(float)i);
    }
    for(int i =0;i< QUEUE_MAX_SIZE;i++){
        float value;
        queue_dequeue(&queue,&value);
        printf("%f\n",value);
    }

    return 0;
}
```

学习记录

markdown平时也在使用，常用的基本掌握了，遇到不知道的还是查手册比较好；栈和队列大部分在上数据结构的课上时都学过差不多了，但当时没有要求我们用C语言去实现结构，只是简单提了一下，现在相当于再次加深对其理解吧
今天中主要是
对C语言进行了一些补充学习，例如:

- 动态申请空间的地址、
- mecpy的使用、
- 函数的调用等等
  熟悉使用VSCode、CMake