动态数组——API思路详解

写在前面：这里是动态数组API简介：版本是v1.0.0!这篇API博客我写了4个小时，匆匆忙忙。但却是我第一个过一万字的博客！

我第一次手搓大型的小项目，也是我第一次单个程序突破1000行！这个前言就算小小的纪念一下这个成就吧！

鄙人编程年龄仅仅就7个月，还是一枚小白，这个项目里面肯定还有许多不足！如果你对它感兴趣！可以到我的GitHub上下载，测试！如果发现了Bug，麻烦及时反馈给我！（如果还有解决方案我会直接磕头叫声Daddy（逃））

--

-2023/2.7/21：47

动态数组入门

首先，作为入门级别的数据结构，我们毫无疑问的：这个数据结构必须满足增删查改！其中，增加就必须要有一个一个增加的，还有直接加入一大群的，删除也是如此。此外，这个数组可以查询是否有目标元素存在，这就需要我们也写一些相关的函数来支持我们的操作！就是这些，我们还可以为了使我们的打印更加清晰，（有点难过的是：C语言必须多增加一个参数来存放它，使得看起来相当的拥挤），可以自定义打印方式。事实上，这就是我们C++里面的vector容器（我就是想要复刻API哈哈哈）

我们的数据结构首先已经被框死：他是一个数组，存储的是相同类型的元素：要不全是整形，要不全是字符，要不全是指针，要么就是同一个类型的结构体！于是，我们不妨这样设想：他必须还是数组，那就说明内存的物理结构还是连续的，但是又要不停的扩展或者伸缩！那就把我们的数组（严肃的讲是静态数组）做一些改动，使之可以自由扩展不就好了嘛？这就需要我们使用头文件,来引入malloc函数群就好了

```
#include<stdio.h>//IO流交互
#include<stdlib.h>//标准库：里面有我们想要的函数
#include<string.h>//memory函数
```

基于上面的思路，我们很快就搓出来一个这个玩意：

```
int main()
{
    void* data = NULL;
    int current_size=0;
    //...Do something and the size need to increase

    data=realloc(data,current_size+1);
    current_size++;
}
```

不错，这就是我们动态数组的雏形了！不过，这样散架着太捞了，后续也不好维护与处理。那很简单了：结构体用起来嘛！

```
typedef struct _DynamicArray_ {//_HHH_是为了后续智能提示不会产生冲突
    void* DataPiece; //连续内存块
    size_t current_size; //当前的大小如何！不过，使用size_t其实更是说明我们的程序变量是非负的而已！
}DynamicArray;
```

好像也不太好，我们后续要知道，访问起来是void*！这个特殊的指针不能帮我们锁定目标，是一个未确定类型的指针，所以。。。我们不妨加上一个常量，在数据首次入动态数组的时候直接将这样的值初始化！

```
typedef struct _DynamicArray_ {//_HHH_是为了后续智能提示不会产生冲突
    void* DataPiece; //连续内存块
    size_t current_size; //当前的大小如何！不过，使用size_t其实更是说明我们的程序变量是非负的而已！
    size_t total_usable_size;//用来调整数组的大小，同时又允许我们在一定范围内自由插入
    size_t Single_Data_size;//相当于类型名了，这个数据在我们后面十分的重要
}DynamicArray;
```

这样，我想我们的动态数组就具有了极强的通用性：可以像一般的数组那样存储任何数据！

工厂函数与基本调整函数系列

什么是工厂函数？就是产生对象并且返回它供我们程序员使用的函数，比如说，C语言的malloc函数就是一个代表性的工厂函数。他返回一个任意大小的堆空间：

```
(void*)malloc(bit_malloc)//开辟多少字节自己制定，是一个操作性很强的函数
```

我们的函数就必须使用它来开辟我们的内存：但是，直接让用户制定是麻烦的，极易出错的，所以：

默认工厂函数

我们首先需要一个可以产生一个这样的结构体并且还要返回它的函数，为了初始化这样的结构体，我们需要知道我们要开辟多少个元素，元素多大，否则我们不会开辟。函数的原型可以轻而易举的给出：

```
DynamicArray* Init_A_DynamicArray(
    size_t                  expected_number,
    size_t                  datasize
);
```

首先，作为动态数组，灵活性是重要的，用户（也就是咱们），往往压根就不知道我们会有多少个数据入列。于是，我们可能需要自己指定一个默认的大小，一旦我们的数据超过了指标，我们就马上调用一个自己手动写的函数，如你所见，就是这个功能的封装：

```
//...Do something and the size need to increase

    data=realloc(data,current_size+1);// Adjustment
```

来立马调用起来防止指针越界！我们稍后在调整函数里会再次阐述！回到初始化函数$Init_A_DynamicArray$上来！我们首先初始化一个结构体：

```
DynamicArray* pro_usable_space = (DynamicArray*)malloc(sizeof(DynamicArray))
```

如果屏幕前的你有一定编程经验，马上就会意识到如果仅是止步如此会大错特错！因为我们的malloc函数一旦开辟失败就会返回一个NULL指针，如果我们不对它进行空判断就会闹出程序崩溃的笑话，这里我们看到，一般的教程是直接这么写的：

```
if(!pro_usable_space){
    return;
}
```

如果只是随意教教，这样写可以；工程开发中如此书写很不负责任！首先，我们的程序不可以直接return！后续我们的操作一旦再次拿起甚至时使用返回出去的NULL指针，马上又会闹起报错的笑话！既然如此，我们做一点报错友善处理！·

```
if(pro_usable_space == NULL){
    printf("Sorry! Failed to malloc a new space! Program exits!");
    exit(-1);
}
```

这样好些了！我们后续开发还会反复用到，何不直接使用宏定义直接节省反复车轮子的工作呢？还是相信智能提示的好！

```
#define SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE printf("Sorry! Failed to malloc a new space! Program exits!") //宏定义的重要作用之一：代替代码块

#define DynamicArray_ERROR_IN_MALLOCING_SPACE 1 //编写异常程序可以这样写，使之错误码和错误信息完全对应起来

//  Do Something
...

 if(pro_usable_space == NULL)
 {
    SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
    exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
 }
```

随后顺手初始化，回到刚刚的问题，我们需要给数据结构一个默认的大小，我们不妨设置成5！

```
#define Dyarr_DEFAULT_MALLOC 5
```

```
pro_usable_space->DataPiece = NULL;//赋予空指针，防止成为野指针修改其他内存块
pro_usable_space->Single_Data_size = datasize;//初始化单位元素大小
if (expected_number >= Dyarr_DEFAULT_MALLOC)//大于默认的就用户指定的那个大小
    pro_usable_space->total_usable_size = expected_number;
else
    pro_usable_space->total_usable_size = Dyarr_DEFAULT_MALLOC;//顺手处理异常的数字大小
pro_usable_space->current_size = 0;//当下就没有元素在，赋0!
return pro_usable_space;//返回产生的堆区开辟的结构体
```

有的时候我们可能需要直接拿起一个静态数组直接转化，很简单，我们可以这样写：

```
int main()
{
    int arr[10]={1,2,3,4,5,6,7,8,9,10};
    for(int i = 0; i < 10; i++){
        //插入
    }
    //...
}
```

但这样我们需要反复调用函数，导致程序性能很差，我们为什么不直接提供一个接口，一边创造结构体一边入列呢?

升级工厂函数

很简单明了，我们的函数的原型是很容易想到的：你这个数组：是什么？（Single_Data_size），有多少个元素？（current_size），在哪里（Datapiece）的问题！我们产生的结构体还是要返回给用户使用的!

所以，我们的参数需要：数组地址（在哪里），数组元素大小（是什么），数组元素个数（有几个）

```
    DynamicArray* Updata_A_Static_Array_To_Dynamic_Array(
    void*                                               data,
    size_t                                              datasize,
    size_t                                              datanum
    )
```

注意，这个时候要多想！data会不会是空？我们是要看看的！类似的，我们再次构造一个错误码模块：

```
    if (!data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
```

之后的流程相比就很简单了：

```
    DynamicArray* pro_usable_space = MALLOCDYARR(DynamicArray, 1);
    if(!pro_usable_space)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
//。。。
```

Ok，到这里，我们要开始准备初始化了，不同于标准的，默认的工厂函数，我们这里直接就是有数据的！因此，需要直接开始拷贝!

注意：不建议直接把data的地址交给DataPiece直接托管！！！！！！！！！！！之后的我们的操作可能会使数据抹除，但是DataPiece还记得，贸然的直接托管会导致非法访问内存!

所以：我们单独为数据准备一块空间：

```
    void* pro_usable_data = malloc(datasize * datanum*1.5);//预留一些空间
    if(!pro_usable_data)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
```

养成随手判断空的好习惯（乐）。

然后，拿出我们的memcpy，这是因为我们的数据可能啥都是，指定一个类型会使得我们的动态数组丧失通用性！

```
memcpy(
    pro_usable_data,     //dst : 我们的预留空间
    data,               //rsc : 我们的来源：静态数组
    datasize * datanum  //多大？很是显然！个数乘上大小
      );
```

然后初始化

```
    pro_usable_space->DataPiece         = pro_usable_data; //将拷贝的空间交给dataPiece托管
    pro_usable_space->current_size      = datanum;
    pro_usable_space->Single_Data_size  = datasize;
    pro_usable_space->total_usable_size = 1.5 * datanum;//开辟多大就给多大
    return pro_usable_space;
```

复制工厂函数

我们可以模仿C++ STL里面的复制构造：协同上面的升级函数一样，这里不过多的废话！

```
DynamicArray* Init_A_DynamicArray_by_CopyADyarr(
    DynamicArray*                                       dyarr_copied
)
{
    //检查来源是否合法
    if (!dyarr_copied)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }

     //准备空间并检查
    DynamicArray* pro_usable_space = MALLOCDYARR(DynamicArray, 1);
    if (!pro_usable_space)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }

    void* pro_usable_data = MALLOCDYARR(
        char, //一个字节单位
        (dyarr_copied->Single_Data_size * dyarr_copied->current_size)//多大？还是一样嘛！
    );
    if(!pro_usable_space)
    {
```

```
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    //拷贝数据
    pro_usable_space->DataPiece = pro_usable_data;
    memcpy(
        pro_usable_space->DataPiece, //dst : 我们的预留空间
        dyarr_copied->DataPiece, // rsc: 来源：显然是被拷贝的dataPiece
        dyarr_copied->Single_Data_size * dyarr_copied->current_size
    );
    //初始化与托管数据
    pro_usable_space->current_size = dyarr_copied->current_size;
    pro_usable_space->Single_Data_size = dyarr_copied->Single_Data_size;
    pro_usable_space->total_usable_size = dyarr_copied->total_usable_size;
    return pro_usable_space;
}
```

现在，我们的三个基本工厂函数已经做好：可以一览了

```
//1 默认的工厂函数
DynamicArray* Init_A_DynamicArray(
    size_t                                          expected_number,
    size_t                                          datasize
)
{
    DynamicArray* pro_usable_space = MALLOCDYARR(DynamicArray, 1);
    if(!pro_usable_space)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    pro_usable_space->DataPiece = NULL;
    pro_usable_space->Single_Data_size = datasize;
    if (expected_number >= Dyarr_DEFAULT_MALLOC)
        pro_usable_space->total_usable_size = expected_number;
    else
        pro_usable_space->total_usable_size = Dyarr_DEFAULT_MALLOC;
    pro_usable_space->current_size = 0;
    return pro_usable_space;
}

//2.拷贝工厂函数
DynamicArray* Init_A_DynamicArray_by_CopyADyarr(
    DynamicArray*                                   dyarr_copied
)
{
    if (!dyarr_copied)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    DynamicArray* pro_usable_space = MALLOCDYARR(DynamicArray, 1);
    if (!pro_usable_space)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
```

```
    void* pro_usable_data = MALLOCDYARR(char, (dyarr_copied->Single_Data_size * dyarr_copied->current_size));
    if(!pro_usable_space)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    pro_usable_space->DataPiece = pro_usable_data;
    memcpy(pro_usable_space->DataPiece, dyarr_copied->DataPiece, dyarr_copied->Single_Data_size * dyarr_copied->current_size);
    pro_usable_space->current_size = dyarr_copied->current_size;
    pro_usable_space->Single_Data_size = dyarr_copied->Single_Data_size;
    pro_usable_space->total_usable_size = dyarr_copied->total_usable_size;
    return pro_usable_space;
}

//3.升级工厂函数
DynamicArray* Updata_A_Static_Array_To_Dynamic_Array(
    void*                                           data,
    size_t                                          datasize,
    size_t                                          datanum
)
{
    if (!data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    DynamicArray* pro_usable_space = MALLOCDYARR(DynamicArray, 1);
    if(!pro_usable_space)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    void* pro_usable_data = malloc(datasize * datanum*1.5);
    if(!pro_usable_data)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    memcpy(pro_usable_data, data, datasize * datanum);
    pro_usable_space->DataPiece = pro_usable_data;
    pro_usable_space->current_size = datanum;
    pro_usable_space->Single_Data_size = datasize;
    pro_usable_space->total_usable_size = 1.5 * datanum;
    return pro_usable_space;
}
```

现在我们已经有了基本的工厂函数，已经有了一个可以可以操作的对象了！不过，我们还提到，动态数组必须有必要的调整函数，防止大量的插入导致空间不足或者是删除导致大量的空间浪费！

基本调整函数

调整大小，我们首先要拿到希望调整的动态数组，以及我们想要调成多大！函数的原型，显然易见：

```
void Resize_The_Dynamic_Array(
    DynamicArray*                                   dyarr,
    size_t                                          wished_new_space_size
)
```

为了使程序更加健壮，我们还是返回一个NORMAL的FLAG值表示程序逻辑正常，从而更好的调整程序，这里，我先把enum列举列齐：

```
//Error Code Meaning
//Program Normally run till the end return NORMAL as 0
//When Error in mallocing Space the program return 1
//when inputing a NULL we returns -1
//when inputing a invalid input we returns -2
//Locations UnFind we returns -3
//
    typedef enum _DynamicFunctionStatues_ {
        DynamicArray_Normal = 0,//正常返回
        DynamicArray_ERROR_IN_MALLOCING_SPACE=1,//错误1：开辟空间失败
        DynamicArray_NULL_INPUT=-1,//错误-1：传入空值
        DynamicArray_Invalid_Input=-2,//错误-2：  传入不合法的值
        DynamicArray_UnFind=-3, //错误值-3 没有找到（后面没有使用，这里就算遗留了）
        DynamicArray_Invalid_Free=-4//错误值-4 不合法的释放，后面删减的时候会用到
    }DynamicArrayFunctionStatues;
```

以及错误提示宏：

```
#define SHOW_ERROR_DynamicArray_NULL_INPUT printf("\nSorry! Your input NULL!\n")

#define SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE printf("\nSorry!Failed to malloc space for your data\n")

#define SHOW_ERROR_DynamicArray_Invalid_Input printf("\nYour input invalid, reject to run functions\n")

#define SHOW_ERROR_DynamicArray_UnFind printf("\nPositions unfind!\n")

#define SHOW_ERROR_DynamicArray_Invalid_Free printf("\nFree the invalid space,reject to run the functions\n")
```

其他所有的空返回函数全部就会升级成状态标识函数：只需要我们返回的是：DynamicArrayFunctionStatues枚举下的值（有点伤心，如果是C++11以上，我们可以使用Enum 作用域这个概念了）就好了。函数就改成了：

```
DynamicArrayFunctionStatues Resize_The_Dynamic_Array(
    DynamicArray*                                   dyarr,
    size_t                                          wished_new_space_size
)
```

这样，我们的框架又可以搭建起来了：

```
DynamicArrayFunctionStatues Resize_The_Dynamic_Array(
    DynamicArray*                                   dyarr,
    size_t                                          wished_new_space_size
)
{
    //经典检查
    if(!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    ...//Do something
    return DynamicArray_Normal;
}
```

对于调整函数，我们实际上就是在调整数组的堆空间，这下马上就知道函数的核心是什么了！

```
void* newspace = realloc(dataPiece, new_size)
```

不错，就是realloc函数！

```
    //调整
    void* newspace = realloc(
        dyarr->DataPiece,
        wished_new_space_size * (dyarr->Single_Data_size)
    );
    //检查空间是否合法
    if (newspace == NULL)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
```

然后就是托管，和调整结构体参数：

```
    dyarr->DataPiece = newspace;//空间托管
    dyarr->total_usable_size = wished_new_space_size;//调整usable space
```

别跑，还先别返回正常状态，我们想一个问题，万一，我们的函数当前数据大于了可用空间怎么办？既然是就想要这么多，我们何不妨：

```
if (dyarr->current_size > dyarr->total_usable_size)
{
    dyarr->current_size = dyarr->total_usable_size;//调整至正好满溢的状态，后面的数据直接截断归还给操作系统
}//Used when still unable to contain data
```

活干完了，返回正常状态！

```
return DynamicArray_Normal;
```

总体一览工厂函数与基本调整函数：

```
DynamicArrayFunctionStatues Resize_The_Dynamic_Array(
    DynamicArray*                                   dyarr,
    size_t                                          wished_new_space_size
)
{
    if(!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    void* newspace = realloc(dyarr->DataPiece, wished_new_space_size * (dyarr-
>Single_Data_size));
    if (!newspace)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    dyarr->DataPiece = newspace;
    dyarr->total_usable_size = wished_new_space_size;
    if (dyarr->current_size > dyarr->total_usable_size)
        dyarr->current_size = dyarr->total_usable_size;//Used when still unable to contain data
    return DynamicArray_Normal;
}
```

增（加）函数一览

终于，我们可以进行增删查改了！先来看增函数！

增加：一个数据？一堆数据？是往后面加，还是插入式的加？这就引出了四个基本函数：我们先来最简单的追加一个数据！

追加单个

我们仔细问问自己，追加的数据，有多大？玩意要满溢了，要不要调整？OK！我们一个一个按照程序的进行考虑！首先思考函数的原型：

```
DynamicArrayFunctionStatues Push_Back_Into_A_Dynamic_Array(
    DynamicArray*                                   dyarr,
    void*                                           data,
    size_t                                          datasize
);
```

很容易想到吧！我们往哪里插入，插入什么！就是我们的参数列表吧！

好，检查来了：

```
    if (!dyarr && !data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
```

完事了？肯定没有！我们试想一下，用户可能在不知情的情况下，传入一个完全不是本类型的数据进来：怎么办?

```
    if (sizeof(data) != dyarr->Single_Data_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
```

挺好，我们进一步想啊，数据进来了：万一溢出来，怎么办?

别怕，我们不是已经有了调整函数了嘛！调整一下就好了！（不过这里没有用，开销不必要）

现在，数据准备入列！我们准备一块空间：

```
void* Afteradd_piece = realloc(
    dyarr->DataPiece,
    (dyarr->current_size + 1)*dyarr->Single_Data_size
);
if (!Afteradd_piece)
{
    SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
    exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
}
```

入列咯!

```
    //拷贝
    memcpy(
        //注意：我们插入到尾部：那就是第I-1的后面，
        (char*)Afteradd_piece + dyarr->current_size * dyarr->Single_Data_size,
        //拷贝Data
         data,
        //拷贝Single_Data_size大小，或者sizeof(data)，这个随意！
        dyarr->Single_Data_size
    );
    //刷新结构体值
    dyarr->DataPiece = Afteradd_piece;
    dyarr->current_size++;
    //任务完成，返回状态Flag
    return DynamicArray_Normal;
```

尾插多个数据函数1：尾插相同数据

那插入一堆值，也是如此了！我们在尾插的时候，可以开发出来两个函数：一个是尾插一堆一样的值!

```
DynamicArrayFunctionStatues Push_back_Same_data_Into_A_Dynamic_Array(
    DynamicArray*                                   dyarr,
    void*                                           data,
    size_t                                          n_repeat,
    size_t                                          datasize
)
```

一个是尾插一个数组，有点像Update升级函数了：

```
DynamicArrayFunctionStatues Push_Back_Some_data_Into_A_Dynamic_Array(
    DynamicArray*                                   dyarr,
    void*                                           data_array,
    size_t                                          array_num,
    size_t                                          pos,
    size_t                                          data_inarray_size
)
```

一个个来！

```
//检查
    if (!dyarr && !data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (datasize != dyarr->Single_Data_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (dyarr->current_size == dyarr->total_usable_size)
    {
        //先预备好空间，因为大量数据准备入场
        Resize_The_Dynamic_Array(dyarr, 2 * dyarr->total_usable_size);
    }
```

下面是扩展多大空间的问题：我们扩展是按照比例扩展的，那就需要计算比例：再传入调整函数！

```
int datarate = datasize / dyarr->total_usable_size;
if (datarate > 0)
{
    Resize_The_Dynamic_Array(
        dyarr,
        (datarate + 1) * dyarr->total_usable_size
    );
}
```

预留，托管和拷贝：

```
void* newspace = realloc(
    dyarr->DataPiece,
    (dyarr->current_size + n_repeat) * dyarr->Single_Data_size
);
if (!newspace)
{
```

```
    SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
    exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
}
for (int i = 0; i < n_repeat; i++)
{
    //逐个拷贝
    memcpy(
        //插入到后面的第I+cur_size个位置上，第一个插入结束就要插入到后面一个
        (char*)newspace + (dyarr->current_size+i) * dyarr->Single_Data_size,
        data,
        dyarr->Single_Data_size
    );
}
dyarr->DataPiece = newspace;
dyarr->current_size += n_repeat;
return DynamicArray_Normal;
```

尾插一个数组函数

而同类型的数组拷贝推入逻辑上是一样的：但是我们不得不遍历目标数组，再逐个插入

```
DynamicArrayFunctionStatues Push_Back_Some_data_Into_A_Dynamic_Array(
    DynamicArray*                                   dyarr,
    void*                                           data_array,
    size_t                                          array_num,
    size_t                                          pos,
    size_t                                          data_inarray_size
)
{
    if (!dyarr && !data_array)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos<0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (data_inarray_size != dyarr->Single_Data_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    int DataRate = array_num / dyarr->total_usable_size;
    if (DataRate > 0)
    {
        Resize_The_Dynamic_Array(
            dyarr,
            (DataRate + 1) * dyarr->total_usable_size
        );
    }
    void* newspace = realloc(
        dyarr->DataPiece,
        (dyarr->current_size + array_num) * dyarr->Single_Data_size
    );
    if(!newspace)
```

```
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    dyarr->DataPiece = newspace;
    for (int i = 0; i < array_num; i++)
        memcpy(
        //插入到后面的第I+cur_size个位置上，第一个插入结束就要插入到后面一个
            (char*)dyarr->DataPiece + (pos + i) * dyarr->Single_Data_size,
        //这个是数组的，写成&data_array[i]也是无伤大雅的
            (char*)data_array + i * (dyarr->Single_Data_size),
        //拷贝的大小
            dyarr->Single_Data_size
        );
    //调整大小
    dyarr->current_size += array_num;
    return DynamicArray_Normal;
}
```

下面就是有点难度的插入增函数了：我们这样想，我们需要把一个元素插入到目标动态数组的pos位置上。那：使用memmove把想要插入的位置后面的所有数据挪动一个元素大小的位置，那么，目标位置就空出来了一个位置虚以待坐了！前面的工作不再多说了：

插入一个元素函数

函数的原型：

```
DynamicArrayFunctionStatues Insert_Into_A_Dynamic_Array(
    DynamicArray*                                   dyarr,
    void*                                           data,
    size_t                                          pos
)
```

检查：

```
{
    if (!dyarr && !data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos<0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (dyarr->current_size == dyarr->total_usable_size)
        Resize_The_Dynamic_Array(dyarr, 2 * (dyarr->total_usable_size));
    void* Afteradd_piece= realloc(dyarr->DataPiece, (dyarr->current_size + 1) * dyarr-
>Single_Data_size);
    if (!Afteradd_piece)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
```

```
    //。。。
    return DynamicArray_Normal;
}
```

OK，我们开始让后面的数据挪动他们的屁股！

```
memcpy(
    //拷贝到pos+1位置处
        (char*)dyarr->DataPiece + (pos + 1) * dyarr->Single_Data_size,
    //从pos往后的数据
        (char*)dyarr->DataPiece + pos * dyarr->Single_Data_size,
    //大小计算是个数学问题：不过就是那种从i到j有几个数字的问题：j-i+1个！
        dyarr->Single_Data_size * (dyarr->current_size - pos + 1)
    );
```

然后把数据请进来：

```
memmove(
        (char*)dyarr->DataPiece + pos * dyarr->Single_Data_size,
        data,
        dyarr->Single_Data_size
);
dyarr->current_size++;
```

然后嘞：宣布我们干完活了！

```
return DynamicArray_Normal;
```

插入若干数据函数1：插入数组

于是，插入一堆数据，还是一个原理的！

```
DynamicArrayFunctionStatues Insert_Some_Data_Into_A_Dynamic_Array(
    DynamicArray*                                       dyarr,
    size_t                                              pos,
    void*                                               data_array,
    size_t                                              array_num,
    size_t                                              data_inarray_size
)
{
    if (!dyarr && !data_array)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos<0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (!dyarr->current_size)
    {
        Push_Back_Some_data_Into_A_Dynamic_Array(
            dyarr,
            data_array,
```

```
            array_num,
            pos,
            data_inarray_size
        );
        return DynamicArray_Normal;
    }
    if (dyarr->current_size == dyarr->total_usable_size)
        Resize_The_Dynamic_Array(dyarr, 2 * dyarr->total_usable_size);
    int DataRate = array_num / dyarr->total_usable_size;
    if (DataRate > 0)
        Resize_The_Dynamic_Array(dyarr, (DataRate + 1) * dyarr->total_usable_size);
    void* Afteradd_piece = realloc(dyarr->DataPiece, (dyarr->current_size + array_num) * dyarr-
>Single_Data_size);
    if (!Afteradd_piece)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    memmove(
        //预留array_num个位置
        (char*)dyarr->DataPiece + (pos + array_num) * dyarr->Single_Data_size,
        (char*)dyarr->DataPiece + pos * dyarr->Single_Data_size,
        dyarr->Single_Data_size*array_num
    );
    for (int i = 0; i < array_num; i++)
        memcpy(
            //从目标位往后拷贝
            (char*)dyarr->DataPiece + (pos + i) * dyarr->Single_Data_size,
        //这个是数组的，写成&data_array[i]也是无伤大雅的
            (char*)data_array + i * (dyarr->Single_Data_size),
            dyarr->Single_Data_size
        );
    dyarr->current_size += array_num;
    return DynamicArray_Normal;
}
```

插入若干数据：相同数据

还是一样，直接上代码：

```
//About Dynamic Array
//Insert back a lot of data that has a organization that has all the same data
//Use like this:
//insert_back_Same_data_Into_A_Dynamic_Array(dyarr,n_repeat,datasize,pos)
//
DynamicArrayFunctionStatues insert_back_Same_data_Into_A_Dynamic_Array(
    DynamicArray*                                       dyarr,
    void*                                               data,
    size_t                                              n_repeat,
    size_t                                              datasize,
    size_t                                              pos
)
{
    if (!dyarr && !data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
```

```
    }
    if (datasize != dyarr->Single_Data_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (dyarr->current_size == dyarr->total_usable_size)
        Resize_The_Dynamic_Array(dyarr, 2 * dyarr->total_usable_size);
    int datarate = datasize / dyarr->total_usable_size;
    if (datarate > 0)
        Resize_The_Dynamic_Array(dyarr, (datarate + 1) * dyarr->total_usable_size);
    void* newspace = realloc(dyarr->DataPiece, (dyarr->current_size + n_repeat) * dyarr-
>Single_Data_size);
    if (!newspace)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    memmove(
        (char*)dyarr->DataPiece + (pos + n_repeat) * dyarr->Single_Data_size,
        (char*)dyarr->DataPiece + pos * dyarr->Single_Data_size,
        dyarr->Single_Data_size * n_repeat
    );
    for (int i = 0; i < n_repeat; i++)
        memcpy(
            (char*)newspace + (pos + i) * dyarr->Single_Data_size,
            data,
            dyarr->Single_Data_size
        );
    dyarr->DataPiece = newspace;
    dyarr->current_size += n_repeat;
    return DynamicArray_Normal;
}
```

拼接动态数组

欸嘿！我们可不可以拼接一下两个动态数组呢？可以啊！

```
DynamicArrayFunctionStatues AppendByMergeDynamicArray(
    DynamicArray*                                       dyarr_be_appended,
    DynamicArray*                                       exp_append_array
)
```

这下很容易了：我们的dataPiece直接延后exp_append_array->current_size个，再拷贝就完事咯！

```
DynamicArrayFunctionStatues AppendByMergeDynamicArray(
    DynamicArray*                                       dyarr_be_appended,
    DynamicArray*                                       exp_append_array
)
{
    if (!dyarr_be_appended && !exp_append_array)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    //跟插入大量数据是一个道理的!
```

```
    int datarate = exp_append_array->current_size / dyarr_be_appended->total_usable_size;
    if (
        dyarr_be_appended->current_size
        ==
        dyarr_be_appended->total_usable_size
    )
    {
        Resize_The_Dynamic_Array(
            dyarr_be_appended,
            2 * dyarr_be_appended->total_usable_size
        );
    }
    if (datarate > 0)
        Resize_The_Dynamic_Array(dyarr_be_appended, (datarate + 1) * dyarr_be_appended-
>total_usable_size);
    void* AfterAppend = realloc(
        dyarr_be_appended->DataPiece,
        //这一大长串就是两个动态数组一共的大小!
        (dyarr_be_appended->current_size + exp_append_array->current_size) * dyarr_be_appended-
>Single_Data_size
    );
    if (!AfterAppend)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    dyarr_be_appended->DataPiece = AfterAppend;
    memcpy(
        //目标地：被追加的数组的尾部
        (char*)dyarr_be_appended->DataPiece +
        dyarr_be_appended->current_size * dyarr_be_appended->Single_Data_size,
        //源头：被拷贝的数组
        (char*)exp_append_array->DataPiece,
        //大小：被拷贝数组的大小
        exp_append_array->current_size * exp_append_array->Single_Data_size
    );
    //调整大小
    dyarr_be_appended->current_size += exp_append_array->current_size;

    return DynamicArray_Normal;
}
```

一览增（加）函数

OK，我们再一览我们的成果就好了：

```
//About Dynamic Array
//Push back a data into the Dynamic Array
//Use like this: Push_Back_Into_A_Dynamic_Array(dyarr,data)
//
DynamicArrayFunctionStatues Push_Back_Into_A_Dynamic_Array(
    DynamicArray*                                   dyarr,
    void*                                           data,
    size_t                                          datasize
)
{
    if (!dyarr && !data)
```

```
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (datasize != dyarr->Single_Data_size)
    {
        printf("%d ", sizeof(data));
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (dyarr->current_size == dyarr->total_usable_size)
        Resize_The_Dynamic_Array(dyarr, 1.3 * (dyarr->total_usable_size));
    void* Afteradd_piece = realloc(dyarr->DataPiece, (dyarr->current_size + 1)*dyarr->Single_Data_size);
    if (!Afteradd_piece)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    memcpy((char*)Afteradd_piece + dyarr->current_size * dyarr->Single_Data_size, data, dyarr->Single_Data_size);
    dyarr->DataPiece = Afteradd_piece;
    dyarr->current_size++;
    return DynamicArray_Normal;
}

//About Dynamic Array
//Push back a lot of data that has a organization that has all the same data
//Use like this:
//Push_back_Same_data_Into_A_Dynamic_Array(dyarr,data,the_time_data_repeated,sizeof(data))
//
DynamicArrayFunctionStatues Push_back_Same_data_Into_A_Dynamic_Array(
    DynamicArray*                                       dyarr,
    void*                                               data,
    size_t                                              n_repeat,
    size_t                                              datasize
)
{
    if (!dyarr && !data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (datasize != dyarr->Single_Data_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (dyarr->current_size == dyarr->total_usable_size)
        Resize_The_Dynamic_Array(dyarr, 2 * dyarr->total_usable_size);
    int datarate = datasize / dyarr->total_usable_size;
    if (datarate > 0)
        Resize_The_Dynamic_Array(dyarr, (datarate + 1) * dyarr->total_usable_size);
    void* newspace = realloc(dyarr->DataPiece, (dyarr->current_size + n_repeat) * dyarr->Single_Data_size);
    if (!newspace)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
```

```
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    for (int i = 0; i < n_repeat; i++)
        memcpy(
            (char*)newspace + (dyarr->current_size+i) * dyarr->Single_Data_size,
            data,
            dyarr->Single_Data_size
        );
    dyarr->DataPiece = newspace;
    dyarr->current_size += n_repeat;
    return DynamicArray_Normal;
}

//About Dynamic Array
//Push back a lot of data that has a organization that has a organiztion with static array
//Use like this:
//Push_back_Same_data_Into_A_Dynamic_Array(
//dyarr,
//data,
//how_many_data_are_there_in_static_array,
//sizeof(data))
//
DynamicArrayFunctionStatues Push_Back_Some_data_Into_A_Dynamic_Array(
    DynamicArray*                                       dyarr,
    void*                                               data_array,
    size_t                                              array_num,
    size_t                                              pos,
    size_t                                              data_inarray_size
)
{
    if (!dyarr && !data_array)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos<0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (data_inarray_size != dyarr->Single_Data_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    int DataRate = array_num / dyarr->total_usable_size;
    if (DataRate > 0)
        Resize_The_Dynamic_Array(dyarr, (DataRate + 1) * dyarr->total_usable_size);
    void* newspace = realloc(dyarr->DataPiece, (dyarr->current_size + array_num) * dyarr-
>Single_Data_size);
    if(!newspace)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    dyarr->DataPiece = newspace;
    for (int i = 0; i < array_num; i++)
        memcpy(
```

```
            (char*)dyarr->DataPiece + (pos + i) * dyarr->Single_Data_size,
            (char*)data_array + i * (dyarr->Single_Data_size),
            dyarr->Single_Data_size
        );
    dyarr->current_size += array_num;
    return DynamicArray_Normal;
}
// About Dynamic Array
// Insert back a data
// use like this:
// Insert_Into_A_Dynamic_Array(
// dyarr,
// data,
// where_to_insert,
//
DynamicArrayFunctionStatues Insert_Into_A_Dynamic_Array(
    DynamicArray*                                       dyarr,
    void*                                               data,
    size_t                                              pos
)
{
    if (!dyarr && !data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos<0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (dyarr->current_size == dyarr->total_usable_size)
        Resize_The_Dynamic_Array(dyarr, 2 * (dyarr->total_usable_size));
    void* Afteradd_piece= realloc(dyarr->DataPiece, (dyarr->current_size + 1) * dyarr-
>Single_Data_size);
    if (!Afteradd_piece)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    memcpy(
        (char*)dyarr->DataPiece + (pos + 1) * dyarr->Single_Data_size,
        (char*)dyarr->DataPiece + pos * dyarr->Single_Data_size,
        dyarr->Single_Data_size*(dyarr->current_size-pos+1)
    );
    memmove(
        (char*)dyarr->DataPiece + pos * dyarr->Single_Data_size,
        data,
        dyarr->Single_Data_size
    );
    dyarr->current_size++;
    return DynamicArray_Normal;
}

//About Dynamic Array
//Insert back a lot of data that has a organization that has all the same data
//Use like this:
//insert_back_Same_data_Into_A_Dynamic_Array(dyarr,n_repeat,datasize,pos)
```

```
//
DynamicArrayFunctionStatues insert_back_Same_data_Into_A_Dynamic_Array(
    DynamicArray*                                       dyarr,
    void*                                               data,
    size_t                                              n_repeat,
    size_t                                              datasize,
    size_t                                              pos
)
{
    if (!dyarr && !data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (datasize != dyarr->Single_Data_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (dyarr->current_size == dyarr->total_usable_size)
        Resize_The_Dynamic_Array(dyarr, 2 * dyarr->total_usable_size);
    int datarate = datasize / dyarr->total_usable_size;
    if (datarate > 0)
        Resize_The_Dynamic_Array(dyarr, (datarate + 1) * dyarr->total_usable_size);
    void* newspace = realloc(dyarr->DataPiece, (dyarr->current_size + n_repeat) * dyarr-
>Single_Data_size);
    if (!newspace)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    memmove(
        (char*)dyarr->DataPiece + (pos + n_repeat) * dyarr->Single_Data_size,
        (char*)dyarr->DataPiece + pos * dyarr->Single_Data_size,
        dyarr->Single_Data_size * n_repeat
    );
    for (int i = 0; i < n_repeat; i++)
        memcpy(
            (char*)newspace + (pos + i) * dyarr->Single_Data_size,
            data,
            dyarr->Single_Data_size
        );
    dyarr->DataPiece = newspace;
    dyarr->current_size += n_repeat;
    return DynamicArray_Normal;
}

//About Dynamic Array
//Insert back some data that organized in static array
//use like this: Insert_Into_A_Dynamic_Array(
// dyarr,
// where_to_insert,
// data,
// how_many_data_are_there_in_static_array,
// sizeof(data)
//)
//
DynamicArrayFunctionStatues Insert_Some_Data_Into_A_Dynamic_Array(
```

```
    DynamicArray*                                   dyarr,
    size_t                                          pos,
    void*                                           data_array,
    size_t                                          array_num,
    size_t                                          data_inarray_size
)
{
    if (!dyarr && !data_array)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos<0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (!dyarr->current_size)
    {
        Push_Back_Some_data_Into_A_Dynamic_Array(
            dyarr,
            data_array,
            array_num,
            pos,
            data_inarray_size
        );
        return DynamicArray_Normal;
    }
    if (dyarr->current_size == dyarr->total_usable_size)
        Resize_The_Dynamic_Array(dyarr, 2 * dyarr->total_usable_size);
    int DataRate = array_num / dyarr->total_usable_size;
    if (DataRate > 0)
        Resize_The_Dynamic_Array(dyarr, (DataRate + 1) * dyarr->total_usable_size);
    void* Afteradd_piece = realloc(dyarr->DataPiece, (dyarr->current_size + array_num) * dyarr-
>Single_Data_size);
    if (!Afteradd_piece)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    memmove(
        (char*)dyarr->DataPiece + (pos + array_num) * dyarr->Single_Data_size,
        (char*)dyarr->DataPiece + pos * dyarr->Single_Data_size,
        dyarr->Single_Data_size*array_num
    );
    for (int i = 0; i < array_num; i++)
        memcpy(
            (char*)dyarr->DataPiece + (pos + i) * dyarr->Single_Data_size,
            (char*)data_array + i * (dyarr->Single_Data_size),
            dyarr->Single_Data_size
        );
    dyarr->current_size += array_num;
    return DynamicArray_Normal;
}

//About Dynamic Array
//Append another dynamic Array
// use like this:
```

```
// AppendByMergeDynamicArray(dyarr_be_appended, exp_append_array)
//
DynamicArrayFunctionStatues AppendByMergeDynamicArray(
    DynamicArray*                                       dyarr_be_appended,
    DynamicArray*                                       exp_append_array
)
{
    if (!dyarr_be_appended && !exp_append_array)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    int datarate = exp_append_array->current_size / dyarr_be_appended->total_usable_size;
    if (dyarr_be_appended->current_size == dyarr_be_appended->total_usable_size)
        Resize_The_Dynamic_Array(dyarr_be_appended, 2 * dyarr_be_appended->total_usable_size);
    if (datarate > 0)
        Resize_The_Dynamic_Array(dyarr_be_appended, (datarate + 1) * dyarr_be_appended-
>total_usable_size);
    void* AfterAppend = realloc(
        dyarr_be_appended->DataPiece,
        (dyarr_be_appended->current_size + exp_append_array->current_size) * dyarr_be_appended-
>Single_Data_size
    );
    if (!AfterAppend)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    dyarr_be_appended->DataPiece = AfterAppend;
    memcpy(
        (char*)dyarr_be_appended->DataPiece + dyarr_be_appended->current_size *
dyarr_be_appended->Single_Data_size,
        (char*)exp_append_array->DataPiece,
        exp_append_array->current_size * exp_append_array->Single_Data_size
    );
    dyarr_be_appended->current_size += exp_append_array->current_size;

    return DynamicArray_Normal;
}
```

删减函数

就像人生需要做减法一样，我们也要给我们的数据结构提供删减操作！

尾删除函数（删减一个）

不删减多个，不水代码！

很简单，我们只需要size--就好了！之后再次增加的时候会直接覆盖，同时，若是删减过多可以在后续直接调用Resize函数！

```
// About Dynamic Array
// Actually erase the final element!
// use like this:
```

```
// Pop_Out_From_A_Dynamic_Array(dyarr)
//
DynamicArrayFunctionStatues Pop_Out_From_A_Dynamic_Array(
    DynamicArray*                                           dyarr
)
{
    if(!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    dyarr->current_size--;
    return DynamicArray_Normal;
}
```

直接清除函数（直接清空）

一个思路，但是，这次由于是清空，我们直接free掉数据！

```
DynamicArrayFunctionStatues Clear_A_Dynamic_Array(
    DynamicArray*                                           dyarr
)
{
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    dyarr->current_size = 0;
    if (!dyarr->DataPiece)
    {
        SHOW_ERROR_DynamicArray_Invalid_Free;
        printf("VOID DATA SHOULDN'T BE FREE!");
        exit(DynamicArray_Invalid_Free);
    }
    //预留一个位置，便于后期操作！
    void* pro_usable_space= realloc(dyarr->DataPiece, dyarr->Single_Data_size);
    if (!pro_usable_space)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    dyarr->DataPiece=pro_usable_space;
    dyarr->total_usable_size = 1;
}
```

删除一个数据（指定一个位置）

我们怎么删除数组里的一个数据呢？只需要回顾到Insert函数的操作，我们反过来，让pos后面的数据往前移动一个！不久把数据盖上了嘛！随后size--就是我们想要的结果！

```
// About Dynamic Array
// Actually erase a specific data by pos
// use like this
// Delete_A_Specific_Data_From_the_DynamicArray(dyarr,  pos)
//
```

```
DynamicArrayFunctionStatues Delete_A_Specific_Data_From_the_DynamicArray(
    DynamicArray*                                               dyarr,
    size_t                                                      pos
)
{
    //检查
    if(!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos < 0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    //直接拷贝覆盖跑路!
    memcpy(
        (char*)dyarr->DataPiece+pos*dyarr->Single_Data_size,
        (char*)dyarr->DataPiece+(pos+1)*dyarr->Single_Data_size,
        dyarr->Single_Data_size*(dyarr->current_size-pos-1)
        );
    //size--，防止访问到奇怪的数据!
    dyarr->current_size--;
    return DynamicArray_Normal;
}
```

删除一些数据

这个难度稍大一点：关键在于，如果我们删除了大量的数据，我们需要调用Resize函数来释放多余的空间，这样可以优化内存占用!

```
// About Dynamic Array
// Actually erase some specific data by pos
// use like this
// Delete_A_Specific_Data_From_the_DynamicArray(dyarr, beginpos, endpos)
//
DynamicArrayFunctionStatues Delete_Some_Specific_Data_From_the_DynamicArray(
    DynamicArray*                                               dyarr,
    size_t                                                      Beginpos,
    size_t                                                      Endpos
)
{
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (Beginpos < 0 || Beginpos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (Endpos < 0 || Endpos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
```

```
    }
    memcpy(
        (char*)dyarr->DataPiece + Beginpos * dyarr->Single_Data_size,
        (char*)dyarr->DataPiece + (Endpos + 1) * dyarr->Single_Data_size,
        dyarr->Single_Data_size * (dyarr->current_size - Endpos + 1)
    );
    dyarr->current_size -= Endpos - Beginpos + 1;
    //开始调整大小!
    if (dyarr->current_size <= dyarr->total_usable_size / 4 && dyarr->total_usable_size > 5)
    {
        Resize_The_Dynamic_Array(dyarr, dyarr->total_usable_size / 2);
        dyarr->total_usable_size /= 2;
    }
    return DynamicArray_Normal;
}
```

直接删掉整个动态数组的函数:

注意：我们先释放数据，再释放本身！不然会造成内存泄漏!

```
DynamicArrayFunctionStatues Destroy_A_Dynamic_Array(
    DynamicArray*                                           dyarr
)
{
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_Invalid_Free;
        exit(DynamicArray_Invalid_Free);
    }
    free(dyarr->DataPiece);
    free(dyarr);
    return DynamicArray_Normal;
}
```

一览删除函数:

```
// About Dynamic Array
// Actually erase the final element!
// use like this:
// Pop_Out_From_A_Dynamic_Array(dyarr)
//
DynamicArrayFunctionStatues Pop_Out_From_A_Dynamic_Array(
    DynamicArray*                                       dyarr
)
{
    if(!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    dyarr->current_size--;
    return DynamicArray_Normal;
}

// About Dynamic Array
// Actually erase all elements!
```

```
// use like this:
// Clear_A_Dynamic_Array(dyarr)
//
DynamicArrayFunctionStatues Clear_A_Dynamic_Array(
    DynamicArray*                                           dyarr
)
{
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    dyarr->current_size = 0;
    if (!dyarr->DataPiece)
    {
        SHOW_ERROR_DynamicArray_Invalid_Free;
        printf("VOID DATA SHOULDN'T BE FREE!");
        exit(DynamicArray_Invalid_Free);
    }
    void* pro_usable_space= realloc(dyarr->DataPiece, dyarr->Single_Data_size);
    if (!pro_usable_space)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    dyarr->DataPiece=pro_usable_space;
    dyarr->total_usable_size = 1;
}

// About Dynamic Array
// Actually erase all elements! and unable to be used again!
// use like this:
// Destroy_A_Dynamic_Array(dyarr)
//
DynamicArrayFunctionStatues Destroy_A_Dynamic_Array(
    DynamicArray*                                           dyarr
)
{
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_Invalid_Free;
        exit(DynamicArray_Invalid_Free);
    }
    free(dyarr->DataPiece);
    free(dyarr);
    return DynamicArray_Normal;
}
// About Dynamic Array
// Actually erase a specific data by pos
// use like this
// Delete_A_Specific_Data_From_the_DynamicArray(dyarr,  pos)
//
DynamicArrayFunctionStatues Delete_A_Specific_Data_From_the_DynamicArray(
    DynamicArray*                                           dyarr,
    size_t                                                  pos
)
{
    if(!dyarr)
```

```
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos < 0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    memcpy(
        (char*)dyarr->DataPiece+pos*dyarr->Single_Data_size,
        (char*)dyarr->DataPiece+(pos+1)*dyarr->Single_Data_size,
        dyarr->Single_Data_size*(dyarr->current_size-pos-1)
        );
    dyarr->current_size--;
    return DynamicArray_Normal;
}

// About Dynamic Array
// Actually erase some specific data by pos
// use like this
// Delete_A_Specific_Data_From_the_DynamicArray(dyarr, beginpos, endpos)
//
DynamicArrayFunctionStatues Delete_Some_Specific_Data_From_the_DynamicArray(
    DynamicArray*                                           dyarr,
    size_t                                                  Beginpos,
    size_t                                                  Endpos
)
{
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (Beginpos < 0 || Beginpos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (Endpos < 0 || Endpos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    memcpy(
        (char*)dyarr->DataPiece + Beginpos * dyarr->Single_Data_size,
        (char*)dyarr->DataPiece + (Endpos + 1) * dyarr->Single_Data_size,
        dyarr->Single_Data_size * (dyarr->current_size - Endpos + 1)
    );
    dyarr->current_size -= Endpos - Beginpos + 1;
    if (dyarr->current_size <= dyarr->total_usable_size / 4 && dyarr->total_usable_size > 5)
    {
        Resize_The_Dynamic_Array(dyarr, dyarr->total_usable_size / 2);
        dyarr->total_usable_size /= 2;
    }
    return DynamicArray_Normal;
}
```

打印内容与处理其内数据函数系列

打印操作

打印数据是我们常用的与数据交互的操作，我们正是需要这个东西来看看我们的数据如何：注意到，我们打印的可能不是整数，可能不是字符，而是任何东西！那么，我们所作的，只是提供一个函数接口！至于怎么打印，让用户自己看着办！为了方便打字，为了防止用户花式整出来奇形八怪的Print接口，我们统一一个！

```
typedef void(* MyPrint )(void*)
```

这个是函数指针别名的经典写法，如果你不熟悉，可以从这里看：比若说，我要指定一个类型，给他起个别名便于我们后续写程序提醒与应用（工程上常常这么做！提醒自己这个是干嘛的的）

```
typedef int MyInt
```

这就是说：MyInt是我自己起的一个类型，但是本质上就是Int！它可以跟int 起到完全一致的效果！

```
int main()
{
    int a = 1;
    MyInt b = 1;// Legal!
}
```

而这个：

```
typedef void(* MyPrint )(void*);
```

就是一个函数指针（说白了，就是一个接受一个任意数据啥都不返回的函数！）的别名，这样写，你可能会更有感觉：

```
typedef void(*)(void*) MyPrint ;
```

于是，我们的打印函数就有了点可行的想法了！

打印一个数据（指定位置）

这个函数可以和后面的返回位置的查找函数搭配使用！我们的这个函数打印动态数组里的一个元素！函数肯定的：需要一个动态数组！需要位置！需要打印方式！原型直接出来了：

```
DynamicArrayFunctionStatues Print_Specific_Data_In_A_Dynamic_Array(
    DynamicArray*                                       dyarr,
    MyPrint /*就是函数指针，说白了就是打印方式传进来*/          user_print,
    size_t                                              pos
)
```

还是检查：

```
{
    if (!dyarr && !user_print)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
```

```
    }
    if (pos < 0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    ...

}
```

为了让打印好看点，我们加入表示语句：

```
    printf("\nSpecific Data is Shown:\n");
    ...//here we print
    printf("\nSpecific Data Finished Showing!\n");
    return DynamicArray_Normal;
```

对于函数的调用，我们使用（）调用！

```
user_print(
    (char*)(dyarr->DataPiece) + pos * (dyarr->Single_Data_size)
);
```

为什么这样写？注意，我们必须转化void* 为具体的指针来对指针进行加减操作！否则不合法！，而操作程度最小的指针就是char指针！于是，我们使用：

```
(char*)(dyarr->DataPiece) + pos * (dyarr->Single_Data_size)
```

定位到我们想要打印到的位置！之后解引用的方式（决定访问步长，这是指针的一个应用本质）是由用户决定的！

看看成品：

```
// About Dynamic Array
// Actually Print a specific data
// use like this
//Print_Specific_Data_In_A_Dynamic_Array(dyarr, user_print, pos)
//
DynamicArrayFunctionStatues Print_Specific_Data_In_A_Dynamic_Array(
    DynamicArray*                                           dyarr,
    MyPrint                                                 user_print,
    size_t                                                  pos
)
{
    if (!dyarr && !user_print)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos < 0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    printf("\nSpecific Data is Shown:\n");
    user_print((char*)(dyarr->DataPiece) + pos * (dyarr->Single_Data_size));
```

```
    printf("\nSpecific Data Finished Showing!\n");
    return DynamicArray_Normal;
}
```

打印所有数据

我说停！先别开工！我们引入一个enum，让用户传入一个参数来决定要不要更好的打印我们的数据！

```
//A switch that used in better Print in Dynamic Array!
//Often use OPEN to beautify the control table
//
typedef enum _BetterPrintSwitch_for_dyarr_ {
    Dyarr_OPEN = 1,
    Dyarr_CLOSE = 0
}BetterPrintSwitch_for_dyarr;
```

之后？就是遍历咯！

```
// About Dynamic Array
// Actually Print All data
// use like this
// Print_ALL_Data_In_A_Dynamic_Array(dyarr,user_print_funtional_pointer, Dyarr_OPEN or
Dyarr_ClOSE)
//
DynamicArrayFunctionStatues Print_ALL_Data_In_A_Dynamic_Array(
    DynamicArray*                                           dyarr,
    MyPrint                                              user_print,
    BetterPrintSwitch_for_dyarr                    whether_better_print
)
{
    if (!dyarr && !user_print)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    //要不要更好的打印？
    if(whether_better_print==Dyarr_OPEN)
    {
        printf("\nStart Printing\n");
    }
    //遍历！
    for (int i = 0; i < dyarr->current_size; i++)
    {
        //函数名跟数组名一样！本质上还是指针，所以，无所谓解不解引用
        (*user_print)(
            //打印第I个
            (char*)(dyarr->DataPiece) + i * (dyarr->Single_Data_size)
        );
    }
    if (whether_better_print == Dyarr_OPEN)
    {
        printf("\nFinish Printing\n");
    }
    return DynamicArray_Normal;
}
```

处理数据（单个多个）

仿照Print思路，这一次，我们使用Do_Specific_Change函数指针！

```
typedef void*(*Do_Specific_Change)(void*);//返回void也行，主要考虑扩展接口的问题，后续可能自己定义函数要继续操作
```

这次不在多说！一样的：

```
// About Dynamic Array
// Actually make some change in all data
// use like this
// Do_Specific_Change_To_All_Data(dyarr, user_print, pos)
//
DynamicArrayFunctionStatues Do_Specific_Change_To_All_Data(
    DynamicArray*                                           dyarr,
    Do_Specific_Change                                      user_change
)
{
    if (!dyarr && !user_change)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    for (int i = 0; i < dyarr->current_size; i++)
        user_change(
        //处理第I个数据
        (char*)(dyarr->DataPiece) + i * (dyarr->Single_Data_size)
    );
    return DynamicArray_Normal;
}

// About Dynamic Array
// Actually make some change in a specific data
// use like this
// Do_Specific_Change_To_All_Data(dyarr, user_print, pos)
//
DynamicArrayFunctionStatues Do_Specific_Change_To_Specific_Data(
    DynamicArray*                                           dyarr,
    Do_Specific_Change                                  user_change,
    size_t                                                  pos
)
{
    if (!dyarr && !user_change)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos < 0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    user_change((char*)(dyarr->DataPiece) + pos * (dyarr->Single_Data_size));
    return DynamicArray_Normal;
}
```

打印内容与处理其内数据函数系列一览

```
// About Dynamic Array
// Actually Print a specific data
// use like this
//Print_Specific_Data_In_A_Dynamic_Array(dyarr, user_print, pos)
//
DynamicArrayFunctionStatues Print_Specific_Data_In_A_Dynamic_Array(
    DynamicArray*                                               dyarr,
    MyPrint                                                     user_print,
    size_t                                                      pos
)
{
    if (!dyarr && !user_print)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos < 0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    printf("\nSpecific Data is Shown:\n");
    user_print((char*)(dyarr->DataPiece) + pos * (dyarr->Single_Data_size));
    printf("\nSpecific Data Finished Showing!\n");
    return DynamicArray_Normal;
}
// About Dynamic Array
// Actually Print All data
// use like this
// Print_ALL_Data_In_A_Dynamic_Array(dyarr,user_print_funtional_pointer, Dyarr_OPEN or
Dyarr_ClOSE)
//
DynamicArrayFunctionStatues Print_ALL_Data_In_A_Dynamic_Array(
    DynamicArray*                                                   dyarr,
    MyPrint                                                    user_print,
    BetterPrintSwitch_for_dyarr                           whether_better_print
)
{
    if (!dyarr && !user_print)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    //要不要更好的打印?
    if(whether_better_print==Dyarr_OPEN)
    {
        printf("\nStart Printing\n");
    }
    //遍历!
    for (int i = 0; i < dyarr->current_size; i++)
    {
        //函数名跟数组名一样!本质上还是指针，所以，无所谓解不解引用
        (*user_print)(
            //打印第I个
            (char*)(dyarr->DataPiece) + i * (dyarr->Single_Data_size)
```

```
        );
    }
    if (whether_better_print == Dyarr_OPEN)
    {
        printf("\nFinish Printing\n");
    }
    return DynamicArray_Normal;
}
// About Dynamic Array
// Actually make some change in all data
// use like this
// Do_Specific_Change_To_All_Data(dyarr, user_print, pos)
//
DynamicArrayFunctionStatues Do_Specific_Change_To_All_Data(
    DynamicArray*                                           dyarr,
    Do_Specific_Change                                      user_change
)
{
    if (!dyarr && !user_change)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    for (int i = 0; i < dyarr->current_size; i++)
        user_change(
        //处理第I个数据
        (char*)(dyarr->DataPiece) + i * (dyarr->Single_Data_size)
    );
    return DynamicArray_Normal;
}

// About Dynamic Array
// Actually make some change in a specific data
// use like this
// Do_Specific_Change_To_All_Data(dyarr, user_print, pos)
//
DynamicArrayFunctionStatues Do_Specific_Change_To_Specific_Data(
    DynamicArray*                                           dyarr,
    Do_Specific_Change                                  user_change,
    size_t                                                  pos
)
{
    if (!dyarr && !user_change)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos < 0 || pos>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    user_change((char*)(dyarr->DataPiece) + pos * (dyarr->Single_Data_size));
    return DynamicArray_Normal;
}
```

查询状况与数据函数一览

查询动态数组是否为空函数

呃，最简单了哈哈：不过注意，在C89的时候还没有布尔类型！我们需要自己手搓一个：

```
//Mybool defined
//
typedef enum _bool_ {
    True = 1,
    False = 0
}Bool;
```

用起来咯！

```
// About Dynamic Array
// this is used in judging whether the dynamic array is empty
// use like this:
// isEmpty_InDynamicArray(dyarr)
//
Bool isEmpty_InDynamicArray(
    DynamicArray*                       dyarr
)
{
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (dyarr->current_size==0)
        return True;
    return False;
}
```

查询特定元素是否存在

咳咳，难度来了：对于任意的数据，我们怎么知道他们符不符合要求：相等呢？嘛！这个简单，用户决定嘛！

还是一样，我们统一一个函数指针：其返回值类型是专门用来标记是否存在的：

```
typedef unsigned int Is_Used_Compared_Int;//返回类型
```

```
typedef Is_Used_Compared_Int(*LocateFunc)(void*, void*);
```

这下函数原型好说了！

```
DynamicArrayFunctionStatues isLocateinDyarr(
    DynamicArray*                       dyarr,
    void*                               data,
    LocateFunc                      user_func
)
```

老老老样子，判判判判断！

```
if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (!user_func)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (!data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
```

遍历一下咯！不过为了提高程序可阅读性质：

```
//this type of int is specifically used in Returning Find Or Unfind.
//Recommmend to be used as a nickname:)
//Can use Find or Unfind to return the result
//
typedef enum _DynamicArray_isFind_ {
    Find    = 1,
    Unfind  = -1
}DynamicArray_isFind;
```

```
for (int i = 0; i < dyarr->current_size; i++)
{
    if (
    user_func(
        //check the Ith one
        (char*)dyarr->DataPiece +
        i * dyarr->Single_Data_size,
        data
    )//if found return Find
    )
    {
        return Find;
    }
}
    return Unfind;
```

返回第一次遇到的位置

跟上面很类似捏：

```
DynamicArrayFunctionStatues returnAElembyPos_inDyarr(
    DynamicArray*                                       dyarr,
    void*                                               data,
    LocateFunc                                          user_func
)
{
    if (!dyarr)
    {
```

```
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (!user_func)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (!data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    for (int i = 0; i < dyarr->current_size; i++)
    {
        if ( //check the Ith one
            user_func
            (
                (char*)dyarr->DataPiece + i * dyarr->Single_Data_size,
                data
            )
        )
        {// if found return i(the position)
            return i;
        }
    }
    return Unfind;
}
```

存储位置的动态数组及其API

这个是为了下面做铺垫！原理一致，只展现API！

定义：

```
typedef struct _Dyarrposarr_ {
    int* posSpace;
    size_t pos_size;
}Position_Stored_Dynamic_ArrayFordyarr;
```

更好的打印开关枚举定义：

```
//Want a better Print to beutify your control table? try this one
//
typedef enum _BetterPrintSwitch_for_pos_dyarr_ {
    PSDA_dyarr_OPEN = 1,
    PSDA_dyarr_CLOSE = 0
}BetterPrintSwitch_for_pos_dyarr;
//can use this type when indicating the type
//
typedef unsigned int BetterPrintfor_PosDyarr;
```

API 1：创造一个存储位置的动态数组

```
//About Position_Stored_Dynamic_Array_Func
//Create A Position_Stored_Dynamic_Array! Can init it in this way:
//>>Position_Stored_Dynamic_ArrayFordyarr* Name=Init_A_Postion_Stored_Dynamic_ArrayFordyarr();
//
Position_Stored_Dynamic_ArrayFordyarr* Init_A_Postion_Stored_Dynamic_ArrayFordyarr()
{
    Position_Stored_Dynamic_ArrayFordyarr* pro_usable_space
        =
        MALLOCDYARR(Position_Stored_Dynamic_ArrayFordyarr, 1);
    if (!pro_usable_space)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    pro_usable_space->posSpace = NULL;
    pro_usable_space->pos_size = 0;
    return pro_usable_space;
}
```

API 2：打印所有的位置：

```
//About Positions_Stored_Dynamic_Array
//this function is aimed to output a series of locations that is stored in the pos
////Use PSDA_dyarr_OPEN to have a better shown ,else use PSDA_dyarr_CLOSE
//
DynamicArrayFunctionStatues Show_All_Locations_In_PSDAfor_dyarr(
    Position_Stored_Dynamic_ArrayFordyarr*              posarr,
    BetterPrintfor_PosDyarr                            Whether_Better_Print
)
{
    if (!posarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (Whether_Better_Print == PSDA_dyarr_OPEN)
        printf("\nPositions are shown! :\n");
    for (int i = 0; i < posarr->pos_size; i++)
        printf("%u ", posarr->posSpace[i]);
    if (Whether_Better_Print == PSDA_dyarr_OPEN)
        printf("\nFinish Printing!\n");
    return DynamicArray_Normal;
}
```

API 3：推入一个位置

```
//About Positions_Stored_Dynamic_Array
//this function is aimed to push back a position into the Position_Stored_Dynamic_ArrayFordyarr
//
DynamicArrayFunctionStatues Push_back_a_locations_in_PSDA_dyarr(
    Position_Stored_Dynamic_ArrayFordyarr*              posarr,
    int                                                 pos_acquired
)
{
```

```
    int* pro_usable_space = (int*)realloc(posarr->posSpace, sizeof(int) * (posarr->pos_size +
1));
    if (!pro_usable_space)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    posarr->posSpace = pro_usable_space;
    posarr->posSpace[posarr->pos_size] = pos_acquired;
    posarr->pos_size++;
    return DynamicArray_Normal;
}
```

API4：统计个数

```
//About Positions_Stored_Dynamic_Array
//this function is aimed to return the number of locations
//Use PSDA_dyarr_OPEN to have a better shown ,else use PSDA_dyarr_CLOSE
//
size_t Get_pos_size_From_PSDA_for_dyarr(
    Position_Stored_Dynamic_ArrayFordyarr*              posarr,
    BetterPrintfor_PosDyarr                             Whether_Shown_Print
)
{
    if (!posarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (Whether_Shown_Print == PSDA_dyarr_OPEN)
        printf("\nCurrent targeted data's total num is:%u!\n",posarr->pos_size);
    return posarr->pos_size;
}
```

API 5：销毁一个存储位置的动态数组

```
//About Positions_Stored_Dynamic_Array
//Erase a dynamic Array when donot using it!
//
DynamicArrayFunctionStatues DesTroy_A_PSDA_for_dyarr(
    Position_Stored_Dynamic_ArrayFordyarr*              pointer_to_wishedfreeposarr
)
{
    if (!pointer_to_wishedfreeposarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_Invalid_Free);
    }
    free(pointer_to_wishedfreeposarr->posSpace);
    free(pointer_to_wishedfreeposarr);
    pointer_to_wishedfreeposarr = NULL;
    return DynamicArray_Normal;
}
```

API 6：返回指定位置的标记位置

```
//About Positions_Stored_Dynamic_Array
//return a pos that is targeted!
//
DynamicArrayFunctionStatues getPosbyPosinPSDA(
    Position_Stored_Dynamic_ArrayFordyarr*                  getter,
    size_t                                                 pos
)
{
    if (!getter) {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_Invalid_Free);
    }
    if (pos > getter->pos_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    for (int i = 0; i < pos; i++)
        return (char*)getter->posSpace + i * sizeof(size_t);
}
```

这些API很简单，我不阐述原理了！

返回多个位置函数

返回一个简单，返回多个怎么办？存储位置的动态数组嘛！不过，这个是只记载位置的：那就好说了！

```
Position_Stored_Dynamic_ArrayFordyarr* returnABunchofData_inDyarr(
    DynamicArray*                                   dyarr,
    Position_Stored_Dynamic_ArrayFordyarr*          posArr,
    void*                                           data,
    LocateFunc                                      user_func
)
{
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (!user_func)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (!data)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    for (int i = 0; i < dyarr->current_size; i++)
    {
        for (int i = 0; i < dyarr->current_size; i++)
        {
            if (
                (*user_func)(
```

```
                    data,
                    (char*)dyarr->DataPiece + i * dyarr->Single_Data_size)
            )
            {
                //类似于return i，这次往PSDA里return!
                Push_back_a_locations_in_PSDA_dyarr(posArr, i)
            }
        }
        if (posArr->pos_size!=0)
            return posArr;
        return Unfind;
    }
}
```

交换于排序函数系列

我们的排序只涉及冒泡排序!

交换函数

回忆一下基本的交换方法:

```
void swapInt(int* e1, int*e2){
    int temp = *e1 ;
    *e1 = *e2;
    *e2 = temp;
}
```

基于此:我们还是一样的仿照设计出函数原型:

```
DynamicArrayFunctionStatues swapData_in_DynamicArray(
    DynamicArray*                                       dyarr,
    size_t                                              pos1,
    size_t                                              pos2
)
```

检查:

```
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos1<0 || pos1>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (pos2<0 || pos2>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
```

这一次，我们的交换使用memcpy（memmove都可以）(为了演示我交叉使用了)

```
    //准备空间
    char* swapbit = (char*)malloc(dyarr->Single_Data_size);
    if(!swapbit)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    // int temp = *e1 ;
    memcpy(
        swapbit,
        (char*)dyarr->DataPiece + pos1 * dyarr->Single_Data_size,
        dyarr->Single_Data_size
    );
    //*e1 = *e2;
    memmove(
        (char*)dyarr->DataPiece + pos1 * dyarr->Single_Data_size,
        (char*)dyarr->DataPiece + pos2 * dyarr->Single_Data_size,
        dyarr->Single_Data_size
    );
    // *e2 = temp;
    memcpy(
        (char*)dyarr->DataPiece + pos2 * dyarr->Single_Data_size,
        swapbit,
        dyarr->Single_Data_size
    );
```

别忘了释放空间：

```
    free(swapbit);
    return DynamicArray_Normal;
```

整合看看：

```
// About Dynamic Array
// this is used in swap data
// use like this:
// swapData_in_DynamicArray(dyarr,pos1,pos2)
//
DynamicArrayFunctionStatues swapData_in_DynamicArray(
    DynamicArray*                                           dyarr,
    size_t                                                  pos1,
    size_t                                                  pos2
)
{
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos1<0 || pos1>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
```

```
    }
    if (pos2<0 || pos2>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    char* swapbit = (char*)malloc(dyarr->Single_Data_size);
    if(!swapbit)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    memcpy(
        swapbit,
        (char*)dyarr->DataPiece + pos1 * dyarr->Single_Data_size,
        dyarr->Single_Data_size
    );
    memmove(
        (char*)dyarr->DataPiece + pos1 * dyarr->Single_Data_size,
        (char*)dyarr->DataPiece + pos2 * dyarr->Single_Data_size,
        dyarr->Single_Data_size
    );
    memcpy(
        (char*)dyarr->DataPiece + pos2 * dyarr->Single_Data_size,
        swapbit,
        dyarr->Single_Data_size
    );
    free(swapbit);
    return DynamicArray_Normal;
}
```

冒泡排序交换函数：

仿照经典冒泡排序。。。

```
void bubbleSort(int* arr, int arr_size)
{

    if(arr==NULL){
        return;
    }

    if(arr_size<=1){
        return;
    }

    for(int i=0; i<arr_size-1; i++)
    {
        for(int j = 0;j<arr_size-i-1; j++)
        {
            if(arr[j]> arr[j+1])
            {
                swap(arr[j],arr[j+1]);//swap不实现了，上面就有
            }
        }
    }
```

```
}
```

```
// About Dynamic Array
// sadly that is bubblesort :(
// I haven't make it in quicksort as it was tooooooooooooooooooooo tiring lol
// use like this:
// sort_In_Dynamic_Array(dyarr,comparision_funcuntional_pointer)
//
DynamicArrayFunctionStatues sort_In_Dynamic_Array(
    DynamicArray*                                           dyarr,
    CompareFunc                                             compfunc
)
{
    //检查
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (!compfunc)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    for (int i = 0; i < dyarr->current_size-1; i++)
    {
        for (int j = 0; j < dyarr->current_size - i-1; j++)
        {
            if (//if(arr[j]> arr[j+1])
                compfunc
                        (
                (char*)dyarr->DataPiece + j * dyarr->Single_Data_size,
                (char*)dyarr->DataPiece + (j + 1) * dyarr->Single_Data_size
                        )
                )
                //swap(arr[j],arr[j+1])
                swapData_in_DynamicArray(dyarr, j, j + 1);
        }
    }
}
```

交换于排序函数系列一览

OK，集合一下

```
// About Dynamic Array
// this is used in swap data
// use like this:
// swapData_in_DynamicArray(dyarr,pos1,pos2)
//
DynamicArrayFunctionStatues swapData_in_DynamicArray(
    DynamicArray*                                           dyarr,
    size_t                                                  pos1,
    size_t                                                  pos2
)
{
```

```
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (pos1<0 || pos1>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    if (pos2<0 || pos2>dyarr->current_size)
    {
        SHOW_ERROR_DynamicArray_Invalid_Input;
        exit(DynamicArray_Invalid_Input);
    }
    char* swapbit = (char*)malloc(dyarr->Single_Data_size);
    if(!swapbit)
    {
        SHOW_ERROR_DynamicArray_ERROR_IN_MALLOCING_SPACE;
        exit(DynamicArray_ERROR_IN_MALLOCING_SPACE);
    }
    memcpy(
        swapbit,
        (char*)dyarr->DataPiece + pos1 * dyarr->Single_Data_size,
        dyarr->Single_Data_size
    );
    memmove(
        (char*)dyarr->DataPiece + pos1 * dyarr->Single_Data_size,
        (char*)dyarr->DataPiece + pos2 * dyarr->Single_Data_size,
        dyarr->Single_Data_size
    );
    memcpy(
        (char*)dyarr->DataPiece + pos2 * dyarr->Single_Data_size,
        swapbit,
        dyarr->Single_Data_size
    );
    free(swapbit);
    return DynamicArray_Normal;
}

// About Dynamic Array
// sadly that is bubblesort :(
// I haven't make it in quicksort as it was tooooooooooooooooooooo tiring lol
// use like this:
// sort_In_Dynamic_Array(dyarr,comparision_funcuntional_pointer)
//
DynamicArrayFunctionStatues sort_In_Dynamic_Array(
    DynamicArray*                                           dyarr,
    CompareFunc                                             compfunc
)
{
    if (!dyarr)
    {
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    if (!compfunc)
    {
```

```
        SHOW_ERROR_DynamicArray_NULL_INPUT;
        exit(DynamicArray_NULL_INPUT);
    }
    for (int i = 0; i < dyarr->current_size-1; i++)
    {
        for (int j = 0; j < dyarr->current_size - i-1; j++)
        {
            if (
                compfunc(
                    (char*)dyarr->DataPiece + j * dyarr->Single_Data_size,
                    (char*)dyarr->DataPiece + (j + 1) * dyarr->Single_Data_size
                        )
                )
                swapData_in_DynamicArray(dyarr, j, j + 1);
        }
    }
}
```

测试一下子咯!

测试代码:

```
#define _CRT_SECURE_NO_WARNINGS 1
#include"standard_dynamic_array.h"
MyPrint print(int* e1) {
    printf("%d ", *e1);
}
Do_Specific_Change dochange(int* e1) {
    *e1 = *e1 + 1;
}
LocateFunc loc(int* e1, int* e2) {
    if (*e1 == *e2) {
        return Find;
    }
    return Unfind;
}
int main()
{
    int data1 = 10;

    //test Initing
    DynamicArray* testDyarr1 = Init_A_DynamicArray(10, Dyarr_SIGINT);

    //test a Sigdata pushed:
    Push_Back_Into_A_Dynamic_Array(testDyarr1, &data1);
    Print_ALL_Data_In_A_Dynamic_Array(testDyarr1, print, Dyarr_OPEN);

    //test pushing the same data
    Push_back_Same_data_Into_A_Dynamic_Array(testDyarr1, &data1, 10, Dyarr_SIGINT);
    Print_ALL_Data_In_A_Dynamic_Array(testDyarr1, print, Dyarr_OPEN);

    //Test Clearing
    Clear_A_Dynamic_Array(testDyarr1);
    Print_ALL_Data_In_A_Dynamic_Array(testDyarr1, print, Dyarr_OPEN);
```

```
//test update
int data2[10] = { 1,2,3,4,5,6,7,8,9,10 };
DynamicArray* test2=Updata_A_Static_Array_To_Dynamic_Array(data2, Dyarr_SIGINT, 10);
Print_ALL_Data_In_A_Dynamic_Array(test2, print, Dyarr_OPEN);

//test copy:
DynamicArray* copyone = Init_A_DynamicArray_by_CopyADyarr(test2);
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);

//test insert
printf("before:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);
Insert_Into_A_Dynamic_Array(copyone, &data1, 1);
printf("after:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);

//test insert bunch
printf("before:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);
Insert_Some_Data_Into_A_Dynamic_Array(copyone, 1, data2, 10, Dyarr_SIGINT);
printf("after:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);

//test Erasing sigone
printf("before:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);
Delete_A_Specific_Data_From_the_DynamicArray(copyone, 1);
printf("after:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);

//test Erasing bunch one
printf("before:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);
Delete_Some_Specific_Data_From_the_DynamicArray(copyone,1,10);
printf("after:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);

//test Do change
printf("before:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);
Do_Specific_Change_To_Specific_Data(copyone,dochange,1);
printf("after:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);
printf("before:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);
Do_Specific_Change_To_All_Data(copyone, dochange);
printf("after:");
Print_ALL_Data_In_A_Dynamic_Array(copyone, print, Dyarr_OPEN);

//test location:
DynamicArray* test3 = Updata_A_Static_Array_To_Dynamic_Array(data2, Dyarr_SIGINT, 10);
if (isLocateinDyarr(test3, &data2[2], loc)) {
    printf("\nI have found it!\n");
}
int where = returnAElembyPos_inDyarr(test3, &data2[2], loc);
printf("\nIn dyarr the %d place\n", where);
```

```
    //test Position Stored Dynamic Array and returnBunch
    Clear_A_Dynamic_Array(test3);
    Push_back_Same_data_Into_A_Dynamic_Array(test3, &data2[2], 10, Dyarr_SIGINT);
    Position_Stored_Dynamic_ArrayFordyarr* p1 = Init_A_Postion_Stored_Dynamic_ArrayFordyarr();
    returnABunchofData_inDyarr(test3, p1, &data2[2], loc);
    Print_ALL_Data_In_A_Dynamic_Array(test3, print, Dyarr_OPEN);
    Show_All_Locations_In_PSDAfor_dyarr(p1, PSDA_dyarr_OPEN);
}
```