专业：智 慧 能 源
姓名：陆 天 昊
学号：3190104143
日期：2020/3/11

浙江大学 实验报告

课程名称：C 程序设计专题 指导老师：翁恺 实验名称：Dynamic Array（可变数组）

一、实验要求

编写一个能动态增长的存放int型数据的数组的库。数组的数据类型如下：

```
typedef struct {
    int *content;
    int size;
} Array;
```

该库应提供以下API函数：

```
Array array_create();//返回一个新创建的Array，其中的size为宏BLOCK_SIZE所定义
void array_free(Array* array);//释放Array中的存储空间
int array_size(const Array *array);//返回Array的size
void array_inflate(Array *array);//使得Array增大BLOCK_SIZE个大小
int array_get(const Array *array, int index);//返回在index位置上的int值
void array_set(Array *array, int index, int value);//将index位置上的值置为value
Array array_clone(const Array *array);//复制一个新的Array，其内容与函数参数相同
```

二、实验思路与过程描述

1. **综述：**

传统的数组定义后长度可以看做固定了。定义`int a[40]`这一数组后，虽然可以来一波操作，划另一大块空间，然后让a指向那里。但这样不仅繁琐，而且如此定义`int a[40]`也变得形大于实，40还容易让他人误解变化后数组真实大小。于是封装它，给它简洁的形式很有必要。

2. **其余实验思路与过程均包含在下一块"实验代码解释中"**

三、实验代码解释

`array_create`函数的主要任务便是初始化一个数组的信息——`size`和`content`，对于`content`要分配好内存空间防止其成为野指针；对应的`array_free`便是释放`size`和`content`的信息，用`free(array->content);`

```
Array array_create()
{
    Array a;
    a.size = BLOCK_SIZE;a.content = (int*)malloc(sizeof(int)*a.size);
    return a;
}
```

数组信息的读写`array_get`,`array_set`,`array_size`函数比较简单，调用时返回结构体中对应内容即可。为了实现程序效率的提高，又将程序中只有一行的函数改写成内联函数，以期提高程序的执行效率。

```
static inline int array_size(const Array *array)
    {return array->size;}
static inline int array_get(const Array *array, int index)
    {return array->content[index];}
static inline void array_set(Array *array, int index, int value)
    {array->content[index]=value;}
```

`array_inflate`和`array_clone`稍微复杂些。

1. `array_inflate`

> inflate时需要新开辟一片新空间，大小为`array->size+BLOCK_SIZE`个单位（`int *p=(int*)malloc((array->size+BLOCK_SIZE)*sizeof(int));`），修改好数组的大小。
>
> 再将原有数据一一拷贝至新空间（`for(int i=0;i<array->size;i++)  p[i]=array->content[i];`）
>
> 把这个新空间作为可变数组所指的内容（`array->content=p;`），当然在这之前要把原有空间释放`free(array->content);`否则会导致内存泄漏。
>
> 最终代码如下：

```
void array_inflate(Array *array)
{
    int *p=(int*)malloc((array->size+BLOCK_SIZE)*sizeof(int));
    for(int i=0;i<array->size;i++)
        p[i]=array->content[i];
    array->size=array->size+BLOCK_SIZE;
    free(array->content);
    array->content=p;
}
```

2. `array_clone`

> clone时需要一片新空间（`b.content=(int*)malloc(b.size*sizeof(int));`）
>
> 再将原数组数据一一拷贝至新空间（`for(int i=0;i<array->size;i++) b.content[i]=array->content[i];`）
>
> 克隆数组的内容指针指向这片空间即可。
>
> 这里只能一一拷贝而不能直接`return *array;`是因为该结构体中只有数组内容的地址而不是真的存储了数组内容，所以如此"克隆"两个数组内容指针都指向了同一地方，操作将相互影响。
>
> 最终代码如下：

```
Array array_clone(const Array *array)
{
    Array b;
    b.size=array->size;
    b.content=(int*)malloc(b.size*sizeof(int));
    for(int i=0;i<array->size;i++)
        b.content[i]=array->content[i];
    return b;
}
```

四、实验体会和心得

反思一下这次实验中犯过的错误，最多的就是和`malloc`函数有关。

这首先就体现在`array_inflate`这个函数的构思过程中，当初不理解`int *p=malloc...`它是**把malloc分配出的内存的地址赋给了p**。还天真地想inflate还不简单，想下面两句话了事，结果原来的数据全丢了，还完美造成了**内存泄漏**。lol

```
array->content=malloc((array->size+BLOCK_SIZE)*sizeof(int));
array->size=array->size+BLOCK_SIZE;
```

还有一处，一直潜伏，一直折磨，发现后让我惭愧至极。那是`array_clone`时候，新定义了一个数组之后，`b.content`还是个**野指针**，我就开始`b.content[i]=array->content[i]`。结果导致许多奇奇怪怪症状，自己居然能运作但把后面直接卡死，一上传就是segmentation fault，写出这么一个错误实在让当时的我难以快速定位。

另外，写本题中各种API函数时，很多函数非常简单，**就一两行语句的事**，但是为了**程序可读性**，也把他们**封装**起来。这就让我想到，在寒假里玩STM32单片机的时候，可以直接操作寄存器，但这样写代码，那一定是参考手册不离手，一个一个查这个寄存器这一位是控制什么的，而且要是不写注释那回头再看基本也不认识了。不过官方还把各种功能封装成了函数库，调用函数库看看这个函数名基本就知道这个函数是什么功能了，本来要操作一堆寄存器的初始化过程传一个结构体进去就完事了，学习成本一下就低得多了。通过封装，对程序的**可移植性**也很有利，换平台时只要改动库里面的内容即可，**对外的接口都可以不变**。我玩STM32时调用了一种函数库LL库（Low-layer），其中有很多函数直接操作寄存器，内容也就是一行语句。并且这些函数（如`__STATIC_INLINE void LL_GPIO_SetOutputPin(GPIO_TypeDef *GPIOx, uint32_t PinMask)`）往往前面带`static inline`，查阅后才知道这叫**内联函数**，是以代码膨胀（复制）为代价，而省去了函数调用开销，从而提高函数执行效率。所以对于这些十分短小的函数，还是可以考虑使用内联函数来提高执行效率。

要说心得，我觉得不在程序本身，程序本身说不上难。让我进一步了解了库，并且说来惭愧，这还是我第一次用VSCode，每次编译还得自己打命令调用gcc，还了解到了makefile这个东西。这一下让我觉得，DEV C++简单易用背后也藏住了一些东西，它就像用Arduino，简单直观上手快，但Arduino只是嵌入式的冰山一角，是个玩具，C51到汇编到8051的原理到数电模电……还有很多值得探索的更加基础的东西；C语言也一样，从写代码到了解它的编译器，到操作系统，内核，指令集……。应用层面越来越简单是好事，但我觉得背后得原理，也值得了解。