

图像转ascii大作业分享

👤 1953059 程敬源

01/

分析题目要求

从类的名称即可看出，我们需要实现一个与多维数组相似的类
所以首先参考一下c++中的多维数组

a[0]	a[0][0]	2000 2003	
	a[0][1]	2004 2007	
	a[0][2]	2008 2011	
	a[0][3]	2012 2015	
a[1]	a[1][0]	2016 2019	
	a[1][1]	2020 2023	
	a[1][2]	2024 2027	
	a[1][3]	2028 2031	
a[2]	a[2][0]	2032 2035	
	a[2][1]	2036 2039	
	a[2][2]	2040 2043	
	a[2][3]	2044 2047	

根据上学期的所学内容，多维数组中的数据是以线性方式连续存储的，在本质上是一个一维数组。

在使用时，多维数组可以通过指针地址索引数据

```
int test[2][2][2]={1,2,3,4,5,6,7,8};
```

```
cout<<test;
```

指向原三维数组

```
cout<<test[0];
```

指向一个二维数组

```
cout<<*test[0];
```

指向一个一维数组

```
cout<<**test[0];
```

指向具体元素

```
008FF7F4
```

```
008FF7F4
```

```
008FF7F4
```

```
1
```

多维数组的不足之处

1.定义数组时，各维度大小必须是常量

本题需要根据图像大小建立Array类

2.维数固定，不能方便地进行reshape

本题在读取像素信息时，需要将一维数组转为三维数组以方便索引

3.在作为参数传入/传出时会退化为指针

本题中的Array类需要在不同函数间传递

题目要求

- 1.实现多维数组的基本功能
- 2.在此基础上可以进行reshape
- 3.可以进行基本矩阵运算

Array类中，数据仍然以一维数组方式存储，通过动态内存申请解决大小问题。

Array类的对象起到指针作用，可以分别指向整个数组/某一行/某个元素。

02

Array类的具体实现

Array类中所含变量

Int *data 存储连续数据的首指针

Int index 指向具体数据

Int shape[16] 存储每一维度的大小

Int axisNum 维度总数

Int nowAxis 当前所处维度

```
int* data;  
int index;  
int shape[16];  
int axisNum;  
int nowAxis;
```

对于一个确定数组， data指向的地址一定是不变的
随index的改变， 对象可以指向不同数据
后三项同数组维度有关， 便于进行索引

通过可变参数模板接收数据

所给代码已将数据解包好了，参数数量存储在num中，参数内容存储在list[]中

需要注意的是，建立Array对象时不能出现0个参数的情况，否则数组大小会为0

```
auto num = sizeof...(args);  
size_t list[] = { args... };
```

1.定义一个Array

(1)首先需要有一段空间存放Array中的数据，所以要为data指针申请大小合适的空间

(2)然后计算出同维度相关的量，axisNum等于参数的数量（有几维），shape[16]分别等于各个维度所含的元素数量

(3)该过程中得到的Array对象指向数据起始位置，维度为0(相当于指向整个数组)，令index与nowAxis均等于0

这样我们便得到了一个指向整段数据的指针，作用相当于一个多维数组中的数组名

2.索引

框架中的代码提示我们，进行索引时，函数返回的不是具体数值，而是一个新的Array对象

同指向整段数据的对象不同，这些对象是临时生成的，仅仅用于索引到某一数据

(1) 通过[]进行索引

1.索引到下一个维度：维度数增加1

2.修改数据指向：index增加相应大小

我们能够得到代码：`index+=n*index[++nowAxis]`

(2) 通过at进行索引

1.直接索引至所需位置： nowAxis增加至相应维度

2.修改数据指向： index增加至相应大小

如果前面[]写好的话，此处可以在at函数中多次使用[]进行索引

3.通过reshape改变数组维度

在本过程中，数组存储的数据不会发生变化，所以我们只需改变与维度相关的变量

(1)维度总数会发生变化，需要改变axisNum的大小

(2)每一维度所含的元素数量发生变化，需要重新计算shape[16]中的值

需要注意的是，在改变数组维度时，数据在指针中的一维顺序不会发生变化，所以数据会按照自首至尾的顺序一一对应至新维度上

4.进行矩阵运算

同3相反，在进行矩阵运算时，新矩阵维度与参与运算的两个矩阵相同，但所存储的数据发生变化

我们可以直接通过指针，对底层一维数组中的各个元素进行运算，得到一个新的Array对象并返回

注意函数的安全性

进行reshape时，检测元素数量是否相等

矩阵运算时，保证两个矩阵为同型矩阵

不要忘记添加析构函数

在Array类编写完成后，一定要充分检测各部分功能的可靠性，否则图像处理时会出现许多问题

将Array类应用于读取图像数据

从data中获取的数据是一个大小为 $x*y*4$ 的一维数组，可以将其以Array存储后，转为对应的三维数组，使索引对应像素的数据比较方便。然后通过公式，将RGB数据转化为灰度数据，通过Array存储为大小为 $x*y$ 的二维数组。

(如果在大作业3中，我们还可以为Array类编写一个矩阵分块的函数，将图像划分为 $8*8$ 的小块)

关于vector

本题中所实现的Array类可以对维数进行任意变更，但其大小仍然是固定的

能不能实现一个大小不固定的数组呢？

——STL中的vector

vector也通过线性连续空间存储数据，但在空间用尽时会自动申请更大的内存，并将数据转移到新内存上

通过first、last、end三个指针进行管理

03/

将灰度数据输出为ascii字符

在将灰度数据输出时，首先要考虑灰度与字符的对应关系

题目中为我们提供了一种转换方式：将灰度大致分级，每一级对应一个ascii字符

```
//·将灰度分为15级，每一级由一个ascii字符表示，强度越大ascii字符内容越少(越白) LF
char·asciiStrength[]·=·{·'M', 'N', 'H', 'Q', '$', 'O', 'C', '?', '7', '>', '!', ':', '-', ';', '.', '·'}; LF
LF
//·灰度数据 LF
unsigned·char·grayData[]·=·{·154, 80, 97, 15, 214, 32, 68, 90·}; LF
LF
for·(int·i·=·0;·i·<·8;·i++)·{ LF
····//·255·/·18·=·14, 故不会导致越界且能划分出强度等级 LF
····unsigned·char·asciiIndex·=·grayData[i]·/·18; LF
····cout·<<·asciiStrength[asciiIndex]; LF
}
```

通常情况下，这种映射已经足够好


但是在网络上也可找到一些更细致的映射方式，比如

\$@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\|()1{}[]?-_+~<>i!ll;:, " ^ ` ' .

然后，我们需要考虑输出字符的方式

经过测试，printf/cout速度很慢，不适用于输出字符画
在Demo中提供了许多快速输出字符的函数，其中这个比较合适

```
void testP_RandCF() {  
    /*****  
    * 说明: FastPrinter::setData(const char*, const WORD*)  
    *      该函数用于传入和控制台窗口大小一致的一维数组描述的dataBuffer,  
    *      需要注意的是传入的是一维数组指针  
    *****/  
    FastPrinter printer(120, 50);  
    char* dataBuffer = new char[120 * 50];  
    BYTE* frontColorBuffer = new BYTE[120 * 50 * 3];  
    BYTE* backColorBuffer = new BYTE[120 * 50 * 3];  
  
    for (int i = 0; i < 20; i++) {  
        printer.cleanScreen();  
  
        for (int i = 0; i < 120 * 50; i++) {  
            dataBuffer[i] = rand() % 70 + 48;  
            frontColorBuffer[i * 3] = rand() % 255;  
            frontColorBuffer[i * 3 + 1] = rand() % 255;  
            frontColorBuffer[i * 3 + 2] = rand() % 255;  
            backColorBuffer[i * 3] = rand() % 255;  
            backColorBuffer[i * 3 + 1] = rand() % 255;  
            backColorBuffer[i * 3 + 2] = rand() % 255;  
        }  
  
        printer.setData(dataBuffer, frontColorBuffer, backColorBuffer);  
  
        printer.draw(true);  
    }  
}
```



首先根据图片大小，建立一个大小适当的窗口

函数中的databuff这一维数组存储有需要输出的字符，将灰度值映射为字符后逐个存入

frontcolor与backcolor两个数组是每一字符的前景色与背景色，通常是黑底白字/白底黑字

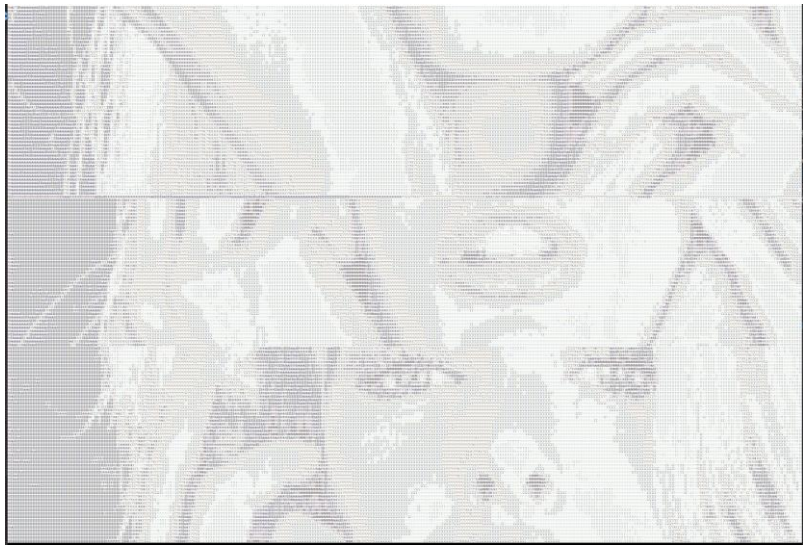
到这里我们的程序已经基本完成，可以对它进行测试了

一点建议

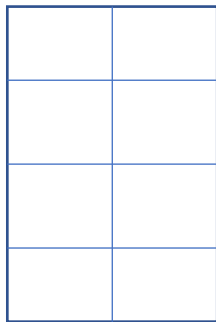
字符画通常很大，我们首先需要将字体调小（在RandCF函数中，调用printer时可设置字体，在1-5之间比较合适），其次可以对图像进行一定压缩

Cmd输出的不是等宽字体，会使图像产生一定变形：可以将长度放大两倍后输出

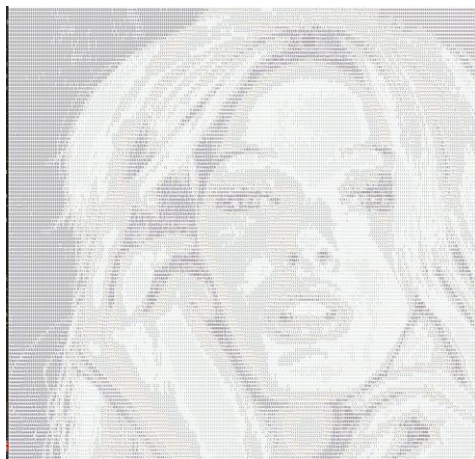
如果出现这种图像错位的情况，一般是输出窗口高度过大，导致RandCF无法正常工作



这时需要通过压缩图片来缩减行数（经测试，高度在100行左右时通常不会发生问题）



我采用了这样一种压缩方式，使用一个字符代表共4行2列的8个像素
这样既可以将高度控制在较小范围内，
又能够使输出的图像是等宽的





THANK YOU

The image features a central rectangular sign with a thick, hand-drawn black border. Inside the sign, the words "THANK YOU" are written in a bold, black, sans-serif font. The sign is set against a white background. In the lower portion of the image, there are several overlapping, semi-transparent pink geometric shapes, including triangles and polygons, creating a layered, abstract effect.