# 二维数组——>指针数组

首先认清数组的性质——一个指向起始地址的指针+一段连续的内存单元

int a[10];

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x00 | 0x04 | 0x08 | 0x12 | 0x16 | 0x20 | 0x24 | 0x28 | 0x32 | 0x36 |

cout<<a<<endl;

将输出起始地址，也就是说a是一个指向起始地址的指针。

Len=10;再向后将造成数组下标越界。因为系统并未为数组开辟这么大的内存。

再看二维数组，二维数组可以考虑成一堆一维数组，每一个数组有一个起始指针，这样二维数组就可以变成一个指针数组。数组中每一个指针元素对应一个一位数组的头指针。

1. int a[1001][1001];
2. int \*a[1001];

使用时无论是二维数组还是指针数组，都是一样的

也就是说只要你的指针指向的内存是有效的，int a[i][j]对于①②两种声明方法，结果是一样的。

为什么要这么写？

Preciously……

int N;

cin>>N;——程序执行时

int A[N];——编译过程中

so……错误！

通常如何？

已知1≤N≤100000

int A[100001];

correct!

然而，当位置数组大小，且实际上n取值较小时，上面这种会造成内存浪费。

因此如何有效地解决这一问题？

动态内存分配。

如何动态地分配数组内存？

使用——指针。

int N;

cin>>N;——程序执行中

int \*A;

A=new int[N];——程序执行中

So……correct！

# 上古卷轴二——动态内存分配

# For C++

# new——动态分配内存

# 使用方法：int \*a=new int[N],其中N为过程中想要动态开辟的大小，可以为变量也可以为常量。

# delete——动态销毁内存

# 使用方法：对于上面int \*a=new int[N];比较理想的是delete[] a，由于int、char、double、float等为基本类型，因此销毁时用delete a;也可。

# For C

# malloc

# #include<malloc.h> or #include<cstdlib>

# int \*a=(int \*)malloc(sizeof(int)\*N);

# free

# 对于上面开辟好内存的数组a，销毁只需要语句free(a);即可实现数组的销毁。