**C与C++中二维数组作为函数参数**

**引言**

如果有一个二维数组，想要以函数的方式打印出来，以下代码是最简单的：

void printMatrix(double data[][5], int rows)

{

    printf("ADDR: %ld\n", data);

    for (int i = 0; i < rows; i++)

    {

        for (int j = 0; j < 5; j++)

        {

            printf("%f  ", data[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}

int main()

{

    double matrix[3][5] = { { -1.0, -2.0, -3.0, -4.0, -5.0 },

    { -6.0, -7.0, -8.0, -9.0, -10.0 },

    { -11.0, -12.0, -13.0, -14.0, -15.0 }

    };

    printMatrix(matrix, 3);

    system("pause");

}

代码能够正确打印出二维数组各个元素，但是，在打印函数中，第一个参数以二维数组的形式传入，二维数组的第二列必须给定，否则不能编译通过，这就限制了打印二维数组的列数，不能适应列数不同的二维数组打印任务，需要考虑其它方法。分析以下几种思路：

**思路一：以二级指针的方式传递二维数组**

以下代码在C++中是不会被编译通过的，在C中，虽然编译通过，但执行时仍会报错：

void putMatrix(double \*\*data, int h, int l)

{

    printf("ADDR: %ld\n", data);

    for (int i = 0; i < h; i++)

    {

        for (int j = 0; j < l; j++)

        {

            printf("%f  ", data[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}

int main()

{

    double matrix[3][5] = { { -1.0, -2.0, -3.0, -4.0, -5.0 },

    { -6.0, -7.0, -8.0, -9.0, -10.0 },

    { -11.0, -12.0, -13.0, -14.0, -15.0 }

    };

    putMatrix(matrix, 3, 5);

    system("pause");

}

putMatrix第一个参数要求的是一个指针的指针变量，而二维数组名为指针数组，即matrix是一个指针数组的首地址，matrix + 1 与 matrix[1] 相同，指向指针数组的第二个成员，示意图如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 指针数组成员 | 各个指针指向的一维数组 | | | | |
| matrix ，matrix[0] | -1 | -2 | -3 | -4 | -5 |
| matrix + 1，matrix[1] | -6 | -7 | -8 | -9 | -10 |
| matrix + 2，matrix[2] | -11 | -12 | -13 | -14 | -15 |

需要注意的是二维数组内存是连续的，因此其本质与一维数组相同，因此可以使用一维数组的寻址方法来打印二维数组。

**思路二：以一维数组的形式进行处理**

由于二维数组内存连续，在内存中，其存储形式与一维数组相同

#define VAL(ARRAY,COLS,H,L) (\*(ARRAY+H\*COLS+L))

**void outputMatrix(double \* data, int h, int l)**

{

    for (int i = 0; i < h; i++)

    {

        for (int j = 0; j < l; j++)

        {

            printf("%f ", VAL(data, l, i, j));

        }

        printf("\n");

    }

    printf("\n");

}

int main()

{

    double matrix[3][5] = { { -1.0, -2.0, -3.0, -4.0, -5.0 },

    { -6.0, -7.0, -8.0, -9.0, -10.0 },

    { -11.0, -12.0, -13.0, -14.0, -15.0 }

    };

**outputMatrix(matrix[0], 3, 5);**

    system("pause");

}

以上代码先定义了一个宏，用于提取二维数组中指定行指定列的元素。在打印函数中，要求输入一个二维数组的指针，因此在主函数中，将二维数组的第一个一维数组的首地址传递进去，由于二维数组内存连续，因此采取该方法相当于一维数组寻址，是没问题的。说明一点：在C中如果直接将二维数组名传入给一维指针参数，也是可以编译通过，并正确执行的，如：

**outputMatrix(matrix, 3, 5);**

但在编译时会报警告信息。而在C++中则不允许这样使用。

**思路三：使用指针的指针来代替二维数组**

C++中定义如下宏：

#define CREATE\_MATRIX(ptr,t,row,col) \

    ptr = new t \*[row];\

    for (int i = 0; i < row; i++)\

    {\

        ptr[i] = new double[col];\

        for (int j = 0; j < col; j++)\

        {\

            ptr[i][j] = 0;\

        }\

    }\

#define  DESTROY\_MATRIX(ptr,row)\

        for (int i = 0; i < row; i++)\

        {\

            delete[] ptr[i];\

            ptr[i] = NULL;\

        }

第一个宏中的t是一个二级指针，该宏首先为其分配了row\*sizeof(\*t)长度的内存空间，其中保存了row个指针，再依次为各指针分配一个col\*sizeof(double)的内存空间。构建完成后内存关系如下图所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ptr |  | \*ptr, ptr[0] | \*ptr+1 | \*ptr+2 | … |
| Ptr+1 |  | Ptr[1],\*(ptr+1) |  |  | … |
| Ptr+2 |  | Ptr[2],\*(ptr+2) |  |  | … |

注意，使用指针的方式对二维数组和二级指针中的元素素进行提取的方式是相同的，例如要取出第二行，第二列的元素，二维数组及二级指针都可以使用\*(\*(data + 1) + 1)或data[1][1]的形式，但是两者的结构不同，在二维数组中，数组名称本身是二维数组的首地址，也是第一个一维数组的首地址，同时也是第一维第一个元素的地址，而二级指针名是一个地址，该地址指向的地址才是第一个一维数组的首地址，也是第一个数组第一个元素的地址，两者的区别如下表所示：

|  |  |
| --- | --- |
| 二维数组 | 二级指针 |
| \*m = m[0] = m，由m指向 | \*m = m[0],由m指向 |
| m[1] = \*(m+1) = m+1，由m+1指向 | m[1] = \*(m+1),由m+1指向 |

可以这么理解以上区别：**二维数组中的m、m+1、m+2保存的是自己的地址，因此对其取\*得到的是自身，而二级指针中m、m+1、m+2保存的是保存数据内存区域的指针，并不是其自身。**

由于C++中，对两者都可以使用相同的寻址形式，因此可以考虑使用二级指针来代替二维数组，而我们却可以使用二维数组的引用方式对其进行操作。

#define CREATE\_MATRIX(ptr,t,row,col) \

    ptr = new t \*[row];\

    for (int i = 0; i < row; i++)\

    {\

        ptr[i] = new double[col];\

        for (int j = 0; j < col; j++)\

        {\

            ptr[i][j] = 0;\

        }\

    }\

#define  DESTROY\_MATRIX(ptr,row)\

        for (int i = 0; i < row; i++)\

        {\

            delete[] ptr[i];\

            ptr[i] = NULL;\

        }

void putMatrix(double \*\*data, int h, int l)

{

    printf("ADDR: %ld\n", data);

    for (int i = 0; i < h; i++)

    {

        for (int j = 0; j < l; j++)

        {

            printf("%f  ", data[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}

int main()

{

    double \*\*data;

    CREATE\_MATRIX(data, double, 3, 5);

    int k = 0;

    for (int i = 0; i < 3;i++)

    {

        for (int j = 0; j < 5;j++)

        {

             data[i][j] = ++k;

        }

    }

    putMatrix(data, 3, 5);

    DESTROY\_MATRIX(data, 3);

    system("pause");

}

在C中，可以这样实现：

#include <stdio.h>

#include <stdlib.h>

#define CREATE\_MATRIX(ptr,t,row,col)    \

    ptr=(t\*\*)malloc(sizeof(t\*)\*(row));  \

    for(j=0;j<(row);j++)    \

            {   \

    ptr[j]=(t\*)malloc(sizeof(t)\*(col)); \

    for(i=0;i<(col);i++)    \

            {   \

    ptr[j][i]=0;    \

            }   \

            }

#define DESTROY\_MATRIX(ptr,row) \

    for(i=0;i<(row);i++)    \

            {   \

    free(ptr[i]);   \

            }   \

    free(ptr);  \

    ptr=NULL;

//在C语言中能够编译通过，但会出错！

void putMatrix(double \*\*data, int h, int l)

{

    for (int i = 0; i < h;i++)

    {

        for (int j = 0; j < l;j++)

        {

            printf("%f  ", data[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}

int main()

{

    int i, j;

    int k = 0;

    double \*\*data;

    CREATE\_MATRIX(data, double, 3, 5);

    for (i = 0; i < 3;i++)

    {

        for (j = 0; j < 5;j++)

        {

            data[i][j] = ++k;

        }

    }

    putMatrix(data, 3, 5);

    DESTROY\_MATRIX(data, 3);

    system("pause");

}