第2章 数组与指针

In [1]:
```
# include <iostream>
# include <cstdio>
# include <string>
using namespace std;
```

数组的初始化

In [2]:
```
int a[] = {1, 2, 3, 4, 5};
char b[] = "hello world";
string c[] = {"zhangsan", "lisi"};
```

数组的访问

In [3]:
```
cout << a[0] << endl;
cout << b[3] << endl;
cout << c[1] << endl;
```

```
1
l
lisi
```

数组名的本质

数组名实际就是一个常指针（常量指针），也就是说这个指针不能更改指向;

由于数组名本质是数组首元素的地址，因此它是一个指向这个首元素的指针。

In [4]:
```
int *p = a;
cout << p[0] << endl;
cout << *(p+1) << endl;
cout << *(a+1) << endl;
```

```
1
2
2
```

数组的大小

比如整数数组a，这里包含5个整数，每个整数占4个字节，因此数组a占20个字节;

对于一个指向数组首地址的指针变量p，它的大小就是这个地址占用空间的大小，8个字节。

In [5]:
```
cout << sizeof(a) << endl;
cout << sizeof(p) << endl;
```

```
20
8
```

二维数组和指针

int p;
这是一个普通的整型变量

int *p;
首先从P 处开始,先与*结合,所以说明P 是一个指针,然后再与int 结合,说明指针所指向的内容的类型为int 型.所以P是一个返回整型数据的指针

int p[3];
首先从P 处开始,先与[]结合,说明P 是一个数组,然后与int 结合,说明数组里的元素是整型的,所以P 是一个由整型数据组成的数组

int *p[3];
首先从P 处开始,先与[]结合,因为其优先级比*高,所以P 是一个数组,然后再与*结合,说明数组里的元素是指针类型,然后再与int 结合,说明指针所指向的内容的类型是整型的,所以P 是一个由返回整型数据的指针所组成的数组

int (*p)[3];
首先从P 处开始,先与*结合,说明P 是一个指针然后再与[]结合(与"()"这步可以忽略,只是为了改变优先级),说明指针所指向的内容是一个数组,然后再与int 结合,说明数组里的元素是整型的.所以P 是一个指向由整型数据组成的数组的指针

int **p;
首先从P 开始,先与*结合,说是P 是一个指针,然后再与*结合,说明指针所指向的元素是指针,然后再与int 结合,说明该指针所指向的元素是整型数据.由于二级指针以及更高级的指针极少用在复杂的类型中,所以后面更复杂的类型我们就不考虑多级指针了,最多只考虑一级指针.

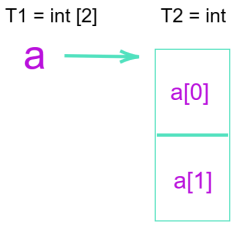
指针定义的基本规则:
假如有一个类型T，定义一个类型T的变量a，就是T a;
如果定义一个指向类型T的指针变量p，那就应该是 T *p;
对于数组，比如int a[3];这里a就是数组类型 int[],

所以定义一个数组应该是 `int[] a;`定义指向这个数组的指针应该是 `int[] (*p);`
但是编译器定义数组的方式并不是把`int[]`看成一个整体`T`，把要定义的变量插在`int`和`[]`中间，
所以我们在理解时，先按`T a`和`T *p`来写出容易理解的形式，然后对于数组，把要定义的`a`或者（`*p`）插到中间即可

例子
参考：https://c.biancheng.net/view/2022.html

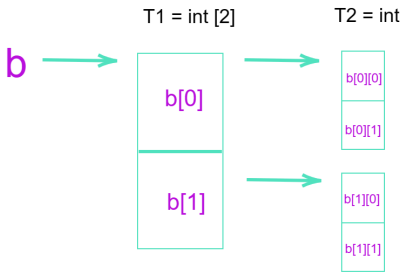
指针与数组（补充）

①. int a[2]



例：
```
int a[2]={23,54};
int *p=a;
```

②. int b[2][2]



例：
```
int b[2][3]={略};
int (*p1)[3]=b;
int *p2[2]={b[0],b[1]};
int *q1=b[0];
int **q2=&q1;
```

In [11]:
```
int matrixA[][3] = {{1, 2, 3}, {4, 5, 6}};
```

第一步：

matrixA指向的就是matrixA[0]的地址，matrixA[0]指向的就是matrixA[0][0]的地址，
matrixA[0]设为T1类型，所以可以 T1 (*p1) = &matrixA[0];*
*matrixA[0][0]设为T2类型，所以可以 T2 (*p2) = &matrixA[0][0];
由于&matrixA[0]就是matrixA，&matrixA[0][0]就是matrixA[0]，所以上面两句替换为：
T1 (*p1) = matrixA;*
*T2 (*p2) = matrixA[0];

第二步：

由于元素matrixA[0]是int[3]类型，元素matrixA[0][0]是int类型，所以上面两句替换为：
int[3] (*p1) = matrixA;*
*int (*p2) = matrixA[0];
调整为正确的格式后如下：

int (*p1)[3] = matrixA;
int (p2) = matrixA[0];

指针指向示意图：

matrixA --> matrixA[0] --> matrix[0][0]
matrixA + 1 --> matrixA[1] --> matrix[1][0]
matrixA指针加1的操作就是加了一个int[3]的空间，也就是12个字节
matrixA[0]指针指向的元素大小为int大小，也就是4个字节
matrixA移动一格就是12个字节，matrixA[0]移动一格就是4个字节

使用指针变量定义后的指向示意图：

p1 --> p2 --> matrix[0][0]
p1 + 1 --> p2 + 3 --> matrix[1][0]

In [12]:
```
int (*p1)[3] = matrixA;
int (*p2) = matrixA[0];
```

输出第一行第二列元素

In [15]:
```
cout << matrixA[0][1] << endl;
cout << *(*p1+1) << endl;
cout << *(matrixA[0]+1) << endl;
cout << *(p2+1) << endl;
```

```
2
2
2
2
```

In [23]:
```
cout << p1[0][1] << endl;
cout << p2[1] << endl;
```

```
2
2
```

输出第二行第二列元素

In [19]:
```
cout << matrixA[1][1] << endl;
cout << *(*(p1+1)+1) << endl;
cout << *(p2+4) << endl;
```

```
5
5
5
```

In [24]:
```
cout << p1[1][1] << endl;
cout << p2[4] << endl;
```

```
5
5
```

存放指针的数组
存放整数的数组是int[],因此存放整型指针的数组是int *[],
所以如下定义：
int p[2]={matrixA[0],matrixA[1]};
matrixA[0]int *p[2]={matrixA[0],matrixA[1]}; ,matrixA[1]正好是两个指针

In [20]:
```
int *p[2]={matrixA[0],matrixA[1]};
```

In [25]:
```
cout << p[1][1];
```

```
5
```

总结：
定义指针按如下形式定义： T (*p); 然后把T换成指定类型即可；
定义数组按如下形式定义：
T a[]; 然后把T换成指定类型即可；

定义一个指向数组的指针，那就是 int[3] (*p); 变成正确形式 int (p) [3];
定义一个存放int*指针的数组，那就是 int* a[2]; 注意a[2]不需要加括号，因为[]优先级高于*

In [ ]:

In [ ]:

In [ ]:
```
const int a = 5;
```

In [ ]:
```
a = 6;
```

In [ ]:
```
int b = a;
b = 6;
cout << a << endl << b ;
```

In [ ]:
```
int arr[]={1,2,3,4};
int *p;
p = arr;
```

In [ ]:
```
struct student{
 char name[20];
 int scores[5];
};
```

In [ ]:
```
student *p;
```

In [ ]:
```
student c={"zhang san",{28,36}}
```

In [ ]:
```
p=&c;
```

In [ ]:
```
cout << c.name;
cout << endl;
cout << p->name;
```

In [ ]:
```
char bbb[]="zhang san";
cout << bbb;
```

In [ ]:
```
int a(6);
```

In [ ]:
```
int a = (5);
```

In [ ]:
```

```