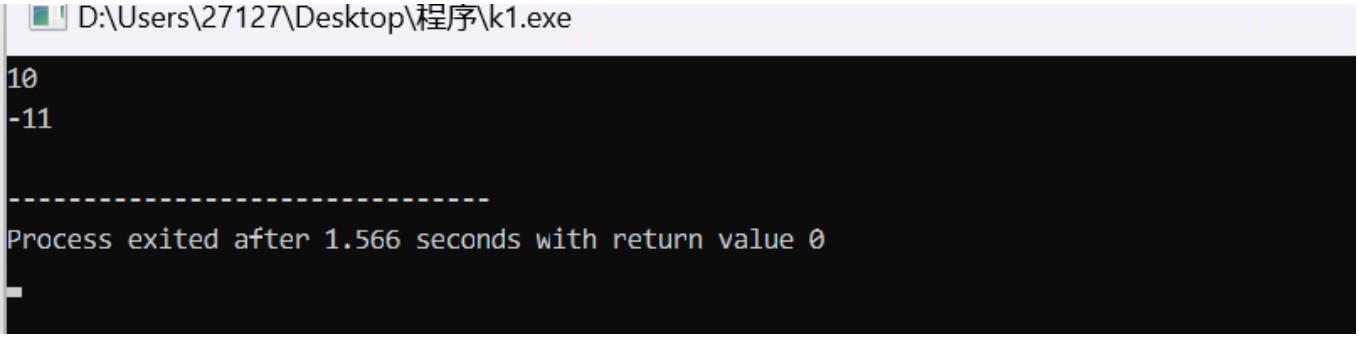
二进制求反

- 将1变成0
- 将0变成1
- 有符号数和无符号数不同，有符号数会导致正负变化，因为最高位是符号位

```c
#include <stdio.h>
int main(){
    int num = 10;
    int num1 = ~num;
    printf("%d\n",num);
    printf("%d\n",num1);

return 0;
}
```



0000 1010
11111 0101

> 对于负数，我们通常使用补码（two's complement）来表示。补码的计算方法如下：
>
> > 首先，找到这个数的反码（one's complement），即将所有的1变为0，所有的0变为1。
> > 然后，给反码加1，得到补码。
> > 对于`11111 0101`：
> >
> > - 反码是`00000 1010`（将所有的1变为0，所有的0变为1）。
> > - 补码是`00000 1011`（给反码加1）。
> >   补码`00000 1011`对应的十进制数是`11`。但是，因为我们最初有一个负数，所以`11111 0101`表示的十进制数是`-11`。
>
> 因此，二进制数`11111 0101`在C中表示的有符号整数是`-11`。

异或运算 ^

Abstract

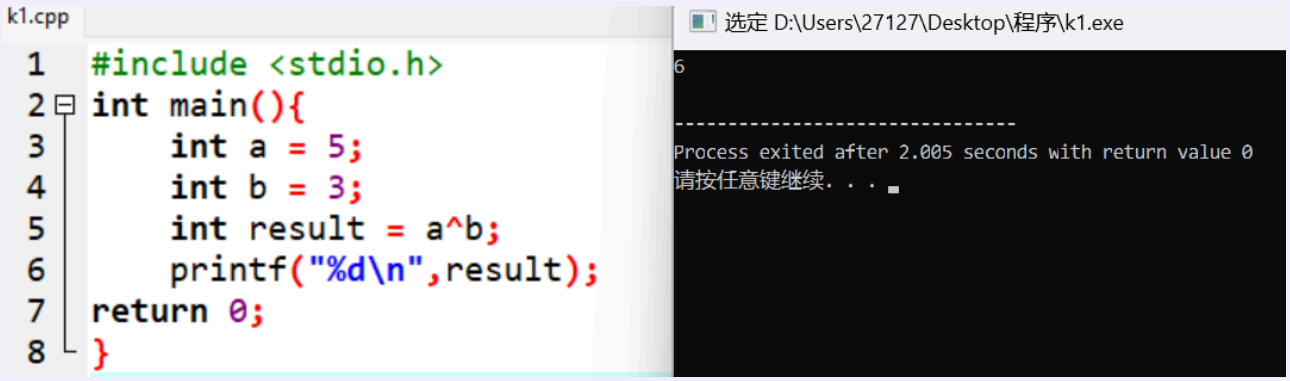
对两个操作数的对应位进行逐位比较，若两个比较的位不同，则结果为1；若相同则结果为0

Important

- 异或具有自反性，自己与自己异或结果是0
- 异或有交换律，且有结合律
- 0与任何数异或是其该数本身

Example

```c
#include <stdio.h>
int main(){
   int a = 5;
 int b = 3;
int result = a^b;
printf("%d\n",result);
return 0;
}
```



0101
0011
异或后为**0110**即为**6**
负数也可以进行异或运算
如-5与5进行异或
1111 1011
0000 0101
异或结果为

1111 1110补码为0000 0010
即为-2

移位运算符

Abstract

将数值的二进制的每一位向左或向右移动一定位数

Caution

左移时，右边用0补齐，左边溢出的数字就被discard，
右移时，无符号整数用0补齐左边，有符号整数用符号位数字补齐左边

Example

```c
#include <stdio.h>
int main()
{
chr a=127;
char b=-128;
char a1=a<<5;
char a2=a>>5;
char b1=b<<5;
char b2 =b>>5;
printf("%d %d %d %d",a1,a2,b1,b2);
return 0;
}
```

返回值为-32 3 0 -4

Caution

如果将char 类型改成int类型那么就要按32位来，char是8位，不过有时候int是16位

Example

```c
#include <stdio.h>

int main()

{

char a=127;

char b=-128;

int a1=a<<5;

int a2=a>>5;

int b1=b<<5;

int b2 =b>>5;

printf("%d %d %d %d",a1,a2,b1,b2);

return 0;

}
```

结果为4064 3 -4096 -4

补码

> 为什么要有补码
>
> - 保持加减法运算规则一致
>   如**-5+3**=-2
>   1111 1011
>   0000 0011
>   加起来是1111 1110
>   1111 1110反码为0000 0001加1即为0000 0010为-2
> - 可以唯一表示0
> - 如-5+5等于1 0000 0000但是那个1是不看的即为0000 0000为0

零扩充和符号扩充

Abstract

将8位整数扩展到32位，把16位扩展到64位等

Attention

零扩充针对非符号数，在左边补0；
符号扩充针对符号数，用符号位进行补充

Example

```c
#include <stdio.h>

int main()

{

    unsigned char u8 =0xFA;

    char i8 = 0xFA;

    unsigned int u32 = u8;

    int i32 = i8;

    printf("u32 = %u\n", u32);

    printf("i32 = %d\n", i32);

    return 0;

}
```

区别 `0x` 和 `\x`

`0x`

- **用途：是一个表示16进制整数常量的前缀**
- **语法：`0x`后面跟上16进制数**

- **表示范围：`0x`用于定义数值类型常量，如`int`，`char`，`long`等，表示该值是十六进制格式的**

\x

- **用途：这是给转义字符，用来在字符串中表示十六进制字节，后面可以跟一个或多个十六进制数字，表示该字节的具体值**
- **语法：同`0x`**

> **Note**
> 补充:
> 二进制的前缀是`0b`,八进制是`0`
> 八进制的转义字符是`\`
> c语言规定:
> `\`后只能跟三位八进制数字
> `\x`后只能跟两位16进制数字

Example

```
```#include <stdio.h>

int main()

{

    int m=0x1AA;

    printf("%d\n",m);

    printf("\x42");

    printf("a\x42a\n");//这是不可以的，会认为/x后面跟3给数字，超出范围

    printf("a\x42%c\n",'a');//因为是字符串的一部分，不需要单引号

    printf("a%ca\n",'\x42');//必须有单引号，这是转义字符，是字符！

    printf("a%da",'\x42');//也有单引号！

    return 0;
```
```

```
}

/*输出结果为
426

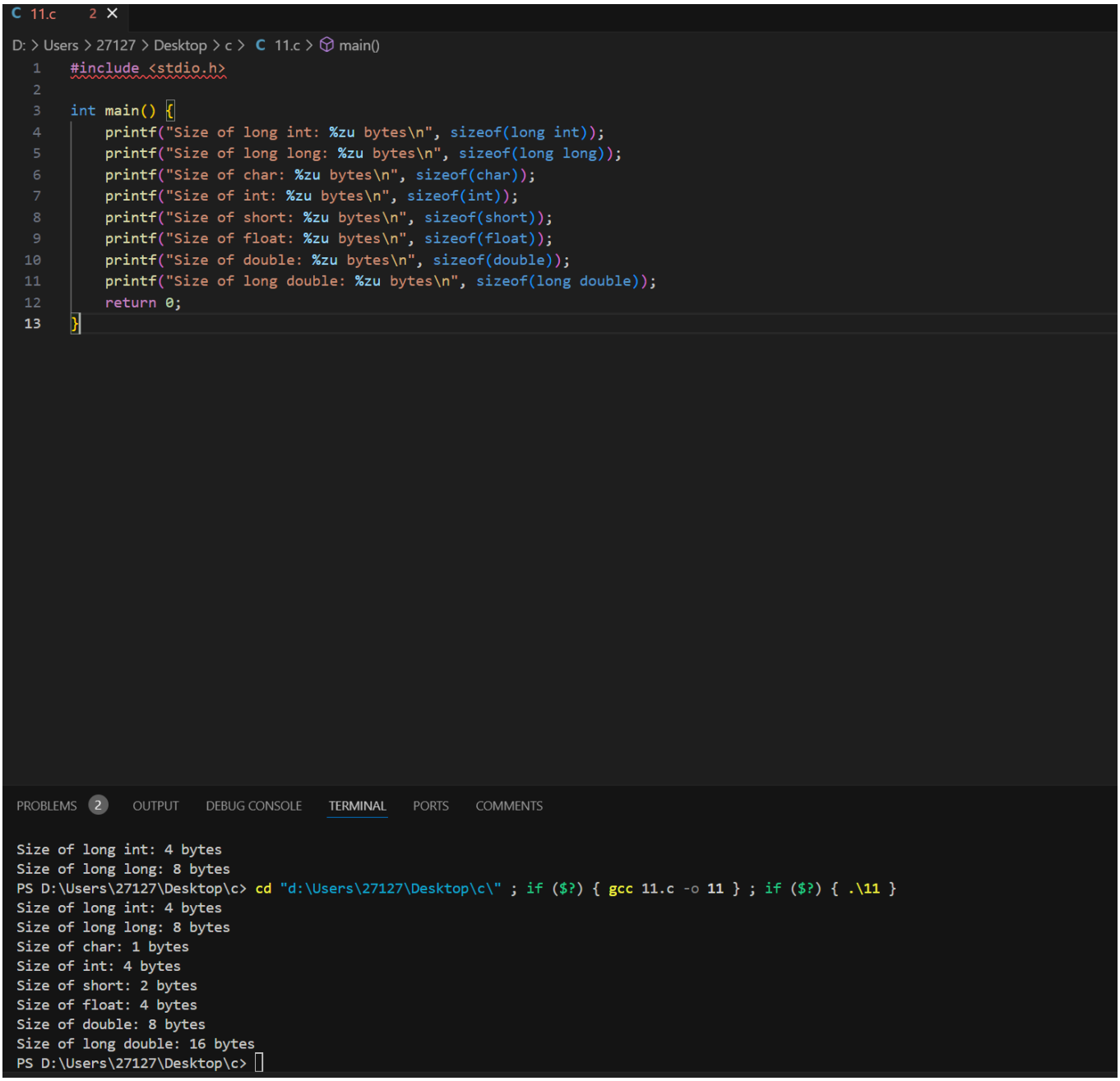
Ba*

aBa

aBa

a66a
*/
```

各种数据类型的大小



数组

```c
#include <stdio.h>

int main()

{

    char s[]={'h','e','y'};

    char a[]="hey";

    char b[]={'a'};//双引号可以不加括号，单引号不可以！
```

```
    char d[]="a";
    for(int i = 0; i < sizeof(s)/sizeof(s[0]); i++) {
        printf("s[%d] = %c\n", i, s[i]);
    }

    for(int i = 0; i < sizeof(a)/sizeof(a[0]) - 1; i++) {
        printf("a[%d] = %c\n", i, a[i]);
    }

    printf("b[0] = %c\n", b[0]);

    printf("d[0] = %c\n", d[0]);
    printf("%s",a);
    printf("%s",b);
}
}
```

Important

```
#include <stdio.h>
int main()
{
    char s[100];
    scanf("%s", &s[0]);
```

```
    //等价于scanf("%s", s);数组名是数组首元素的地址

    printf("%s", s);

    char a[]={'a','\0'};

    printf("%s", a);

    return 0;

}
```

- `scanf` 在敲回车时相当于输入了 `\0`
- 自己创建字符串且不是用“”的时候需要加上斜杠0，否则会出错！因为字符串“hey”本身就已经带了 `\0`
- 只有char类型可以用%s直接输出全部，其他int什么的都要循环输出

```
char str[6] = {'h', 'e', 'l', 'l', 'o', '\0'};  // 手动加上\0
int a[] = {1, 2, 3, 4, 5};//不需要加\0
```

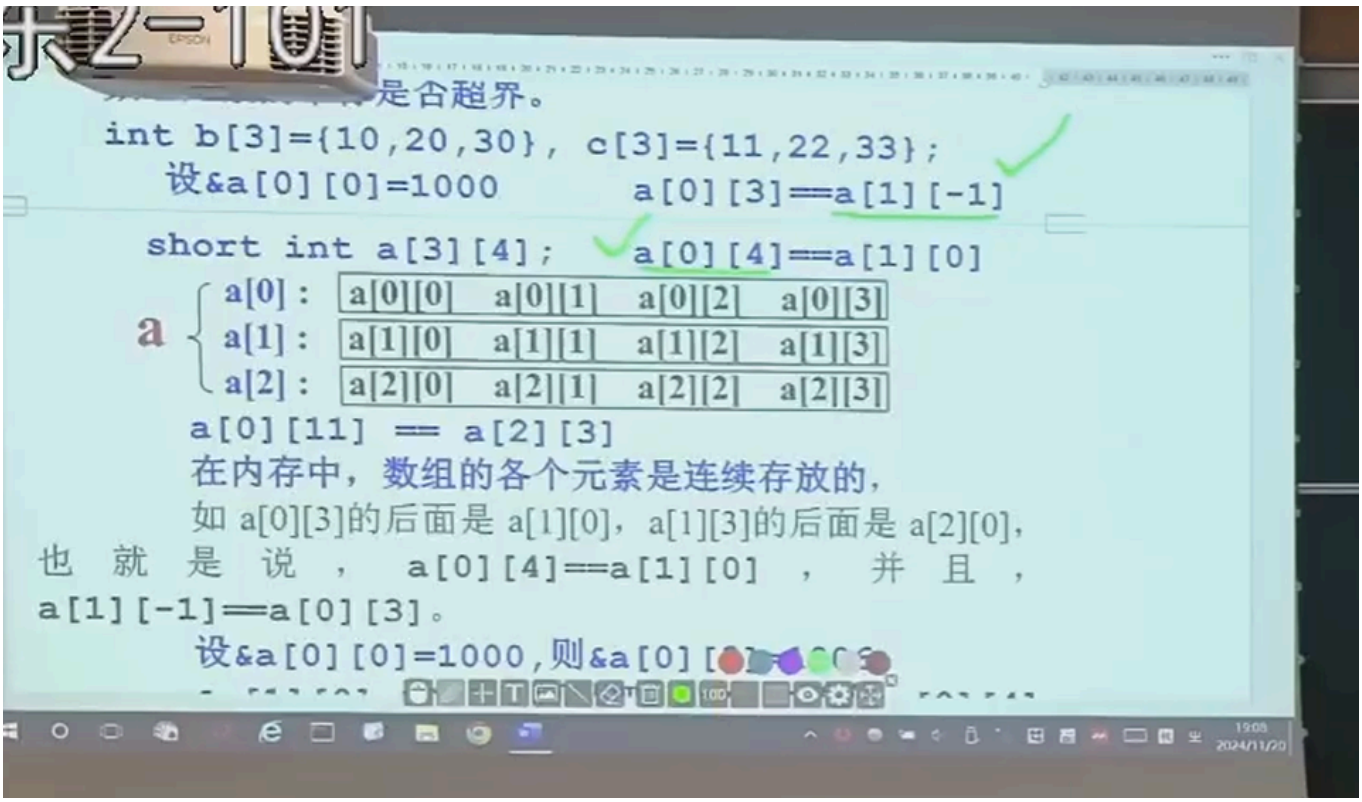
二维数组

```
short int a[3][4];
```

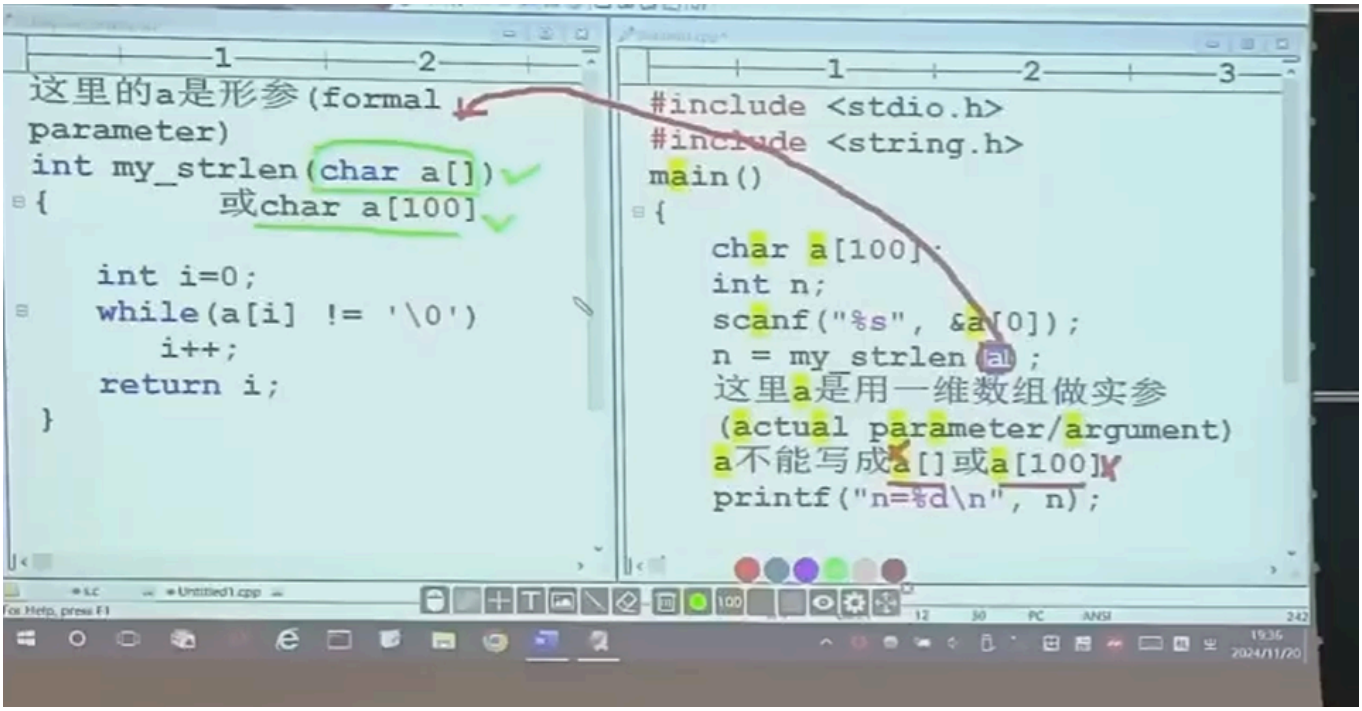
则有a[0] [4]== a[1] [0]

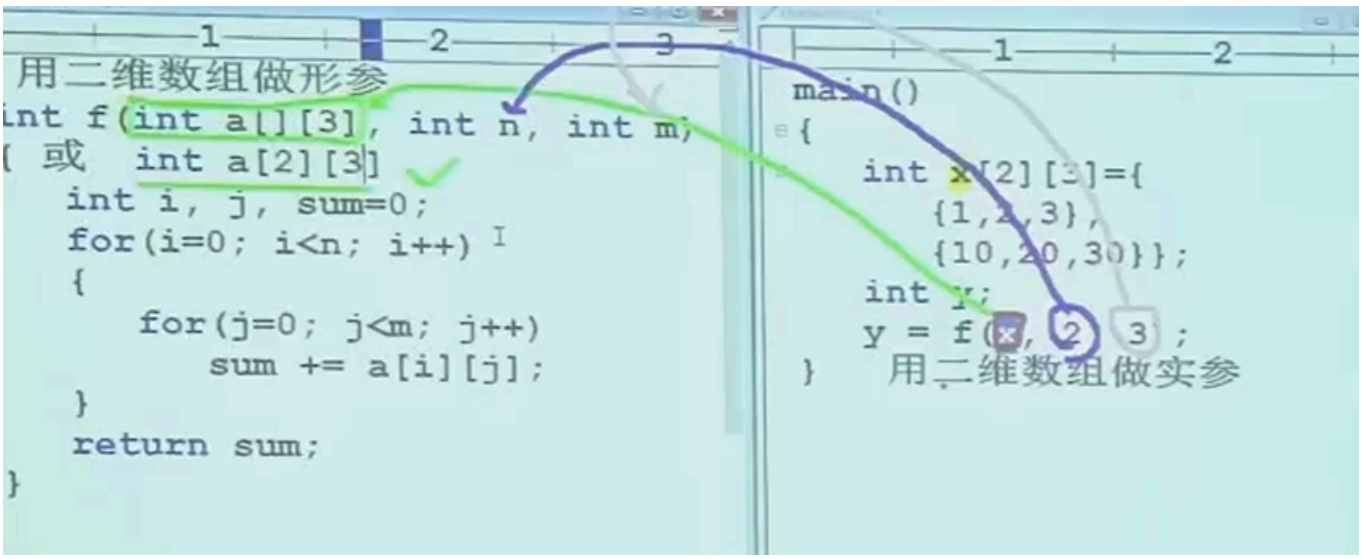

数组元素的下标是否超界。
int b[3]={10,20,30}, c[3]={11,22,33};
设&a[0][0]=1000
a[0][3]==a[1][-1]
short int a[3][4];
a[0][4]==a[1][0]
a
a[0]: a[0][0] a[0][1] a[0][2] a[0][3]
a[1]: a[1][0] a[1][1] a[1][2] a[1][3]
a[2]: a[2][0] a[2][1] a[2][2] a[2][3]
a[0][11] == a[2][3]
在内存中，数组的各个元素是连续存放的，
如 a[0][3]的后面是 a[1][0]，a[1][3]的后面是 a[2][0]，
也就是说，a[0][4]==a[1][0]，并且，
a[1][-1]==a[0][3]。
设&a[0][0]=1000,则&a[0]

越界


是否超界。
int b[3]={10,20,30}, c[3]={11,22,33};
设&a[0][0]=1000
a[0][3]==a[1][-1]
short int a[3][4];
a[0][4]==a[1][0]
a
a[0]: a[0][0] a[0][1] a[0][2] a[0][3]
a[1]: a[1][0] a[1][1] a[1][2] a[1][3]
a[2]: a[2][0] a[2][1] a[2][2] a[2][3]
a[0][11] == a[2][3]
在内存中，数组的各个元素是连续存放的，
如 a[0][3]的后面是 a[1][0]，a[1][3]的后面是 a[2][0]，
也就是说，a[0][4]==a[1][0]，并且，
a[1][-1]==a[0][3]。
设&a[0][0]=1000,则&a[0]

```
short a[][2]={{1,2}, {3,4}, {5,6}, {7,8}};
int n = sizeof(a)/sizeof(a[0]); n=4
n = sizeof(a)/sizeof(a[0][0]); n=8
```

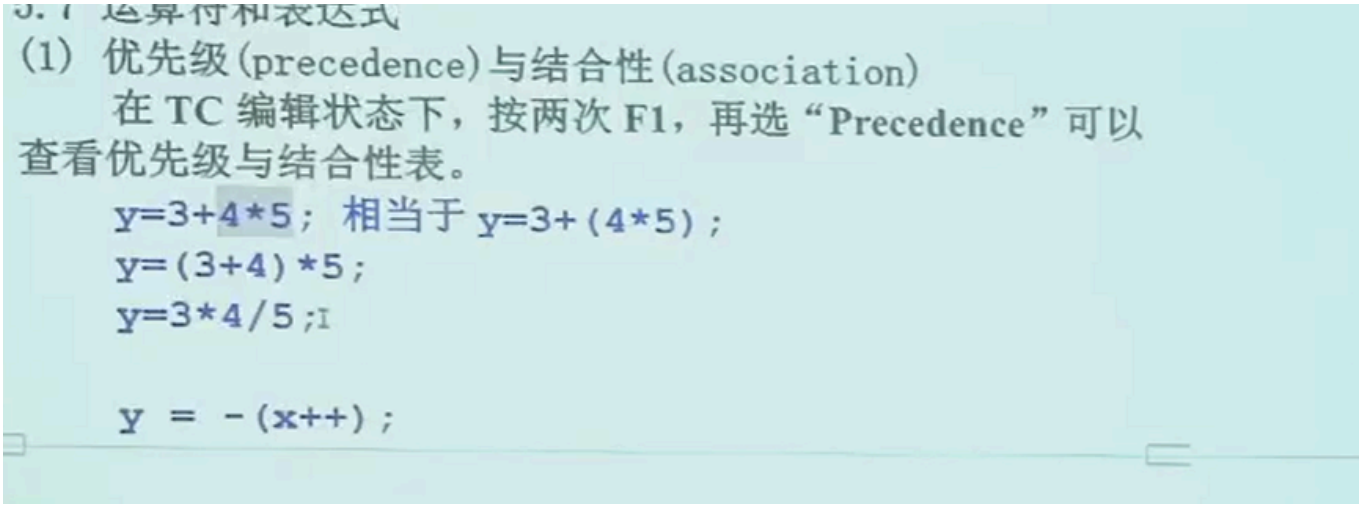




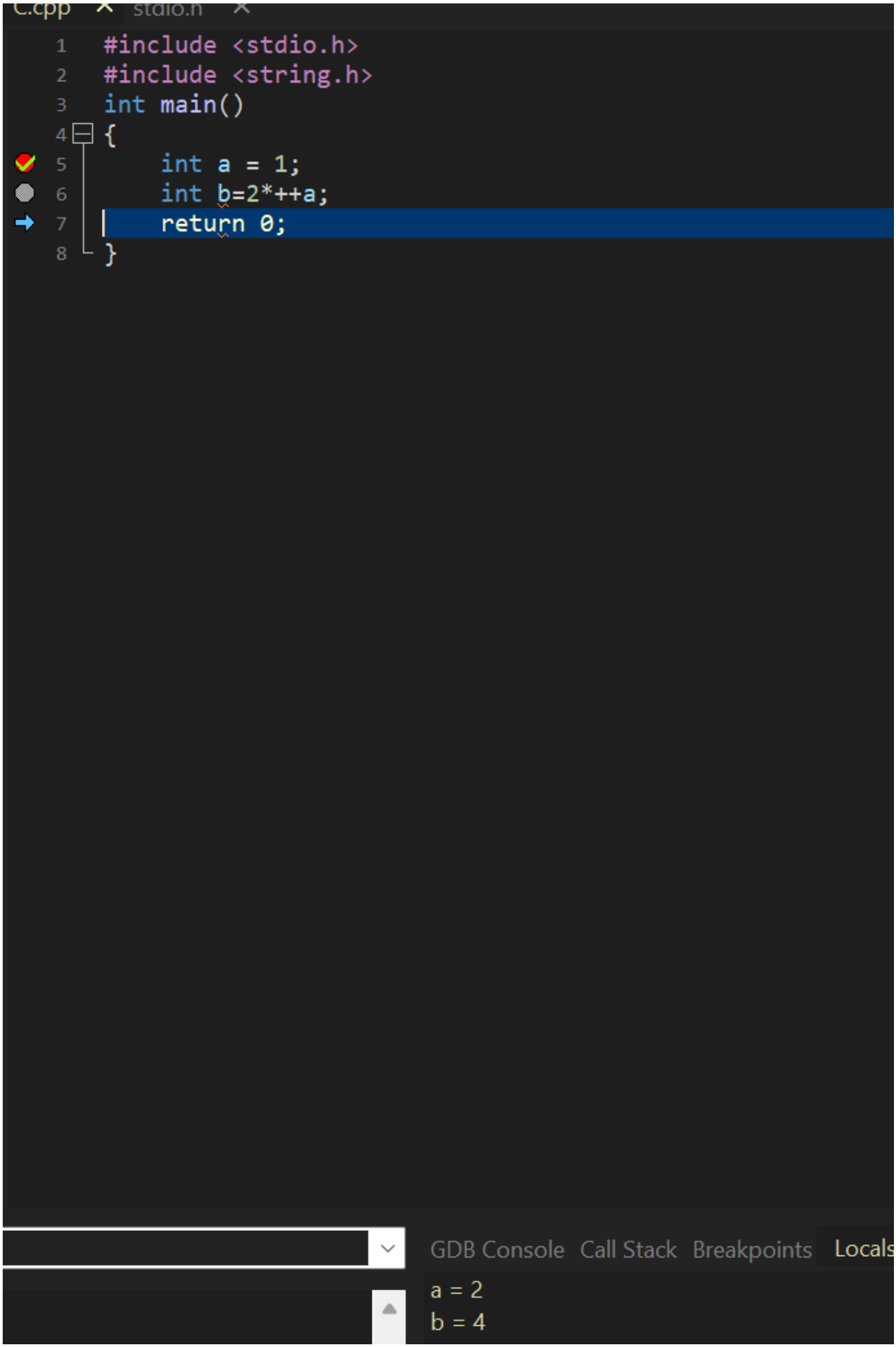
绝对值

- **浮点数** fabs <math.h>
- **整数**abs <stdlib.h>

优先级



left to right左结合
right to left右结合


C.cpp
stdio.h
#include <stdio.h>
#include <string.h>
int main()
{
int a = 1;
int b=2*++a;
return 0;
}
GDB Console
Call Stack
Breakpoints
Locals
a = 2
b = 4

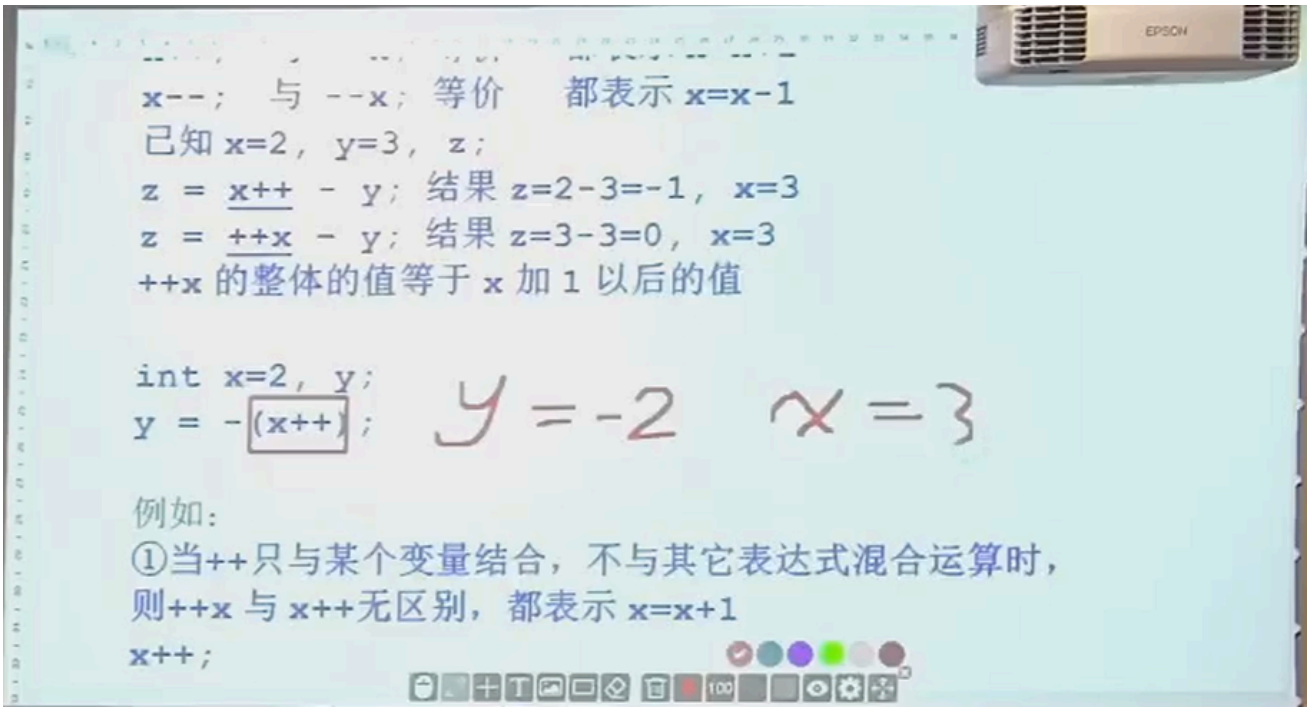
Operator	Order of evaluation
() [] . ->	left to right
! ~ - ++ -- & * (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right

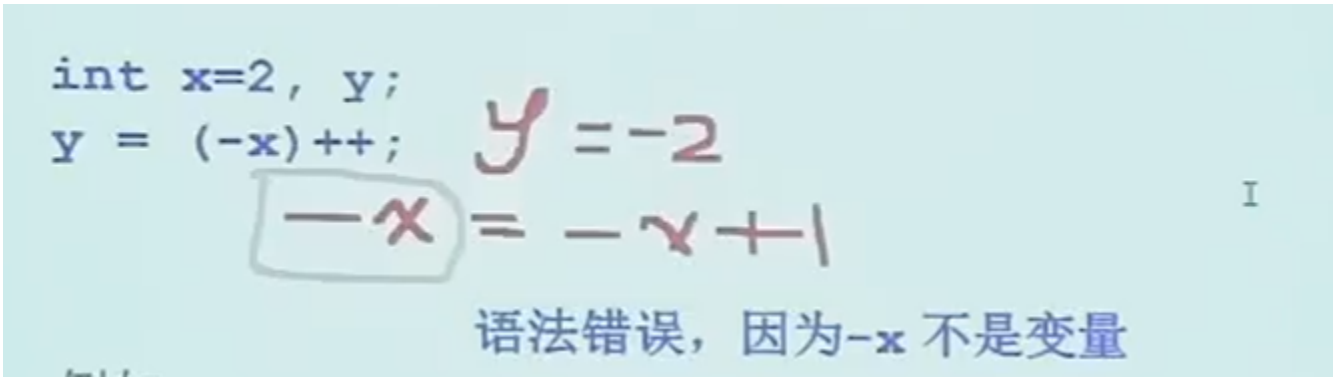
PgDn

Precedence of Operators (cont)

Operator	Order of evaluation
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= etc.	right to left
,	left to right

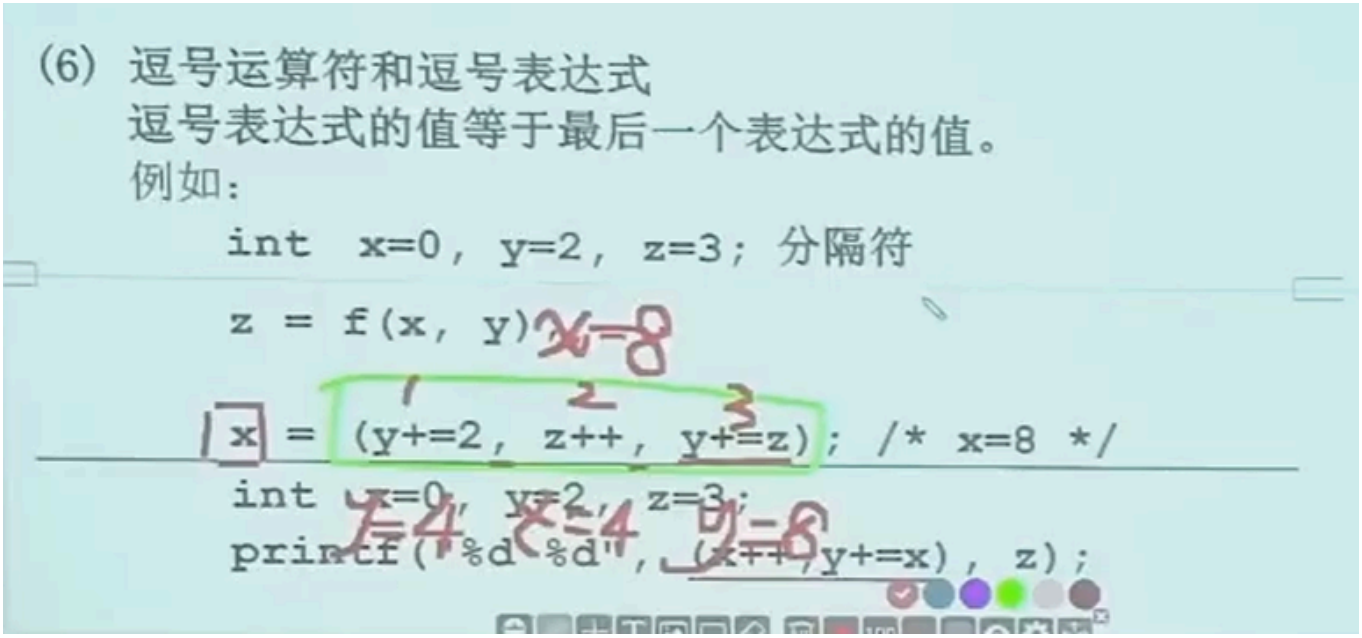
PgUp





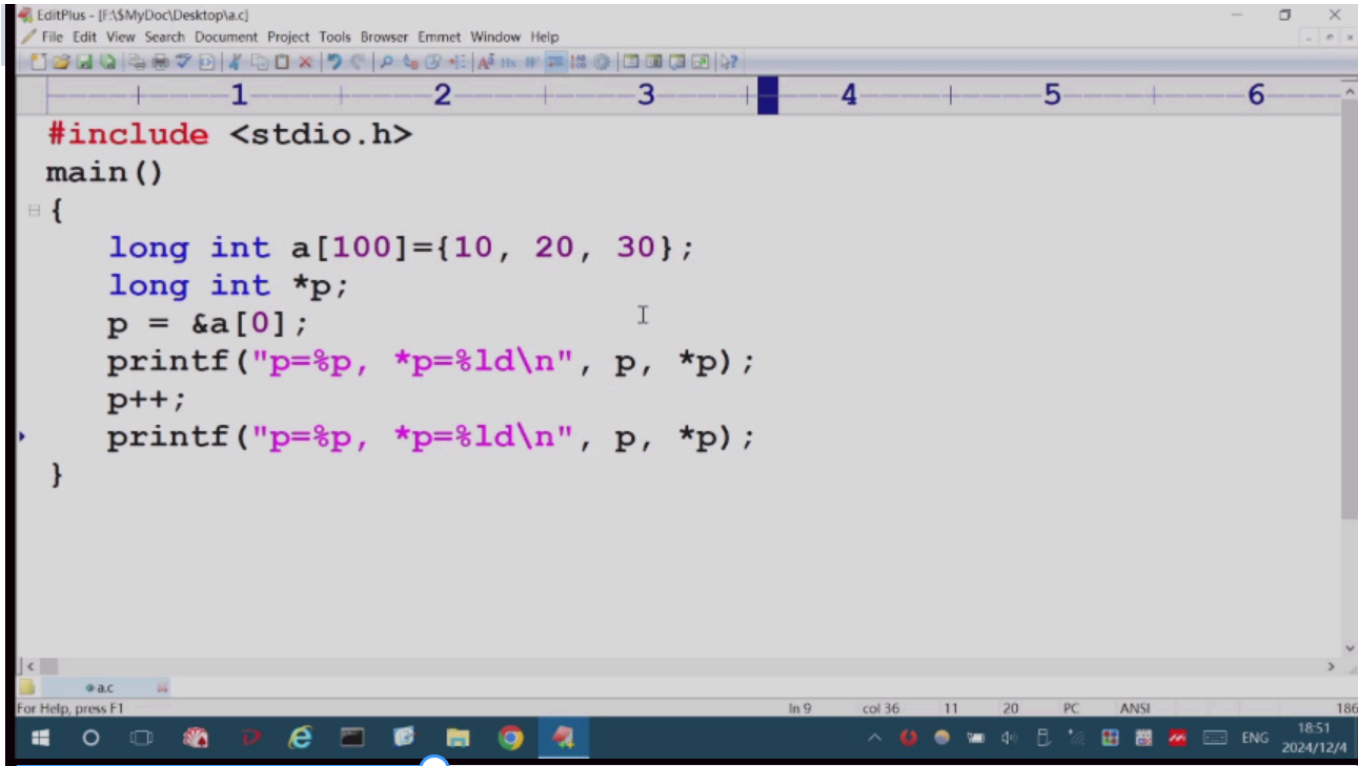
!

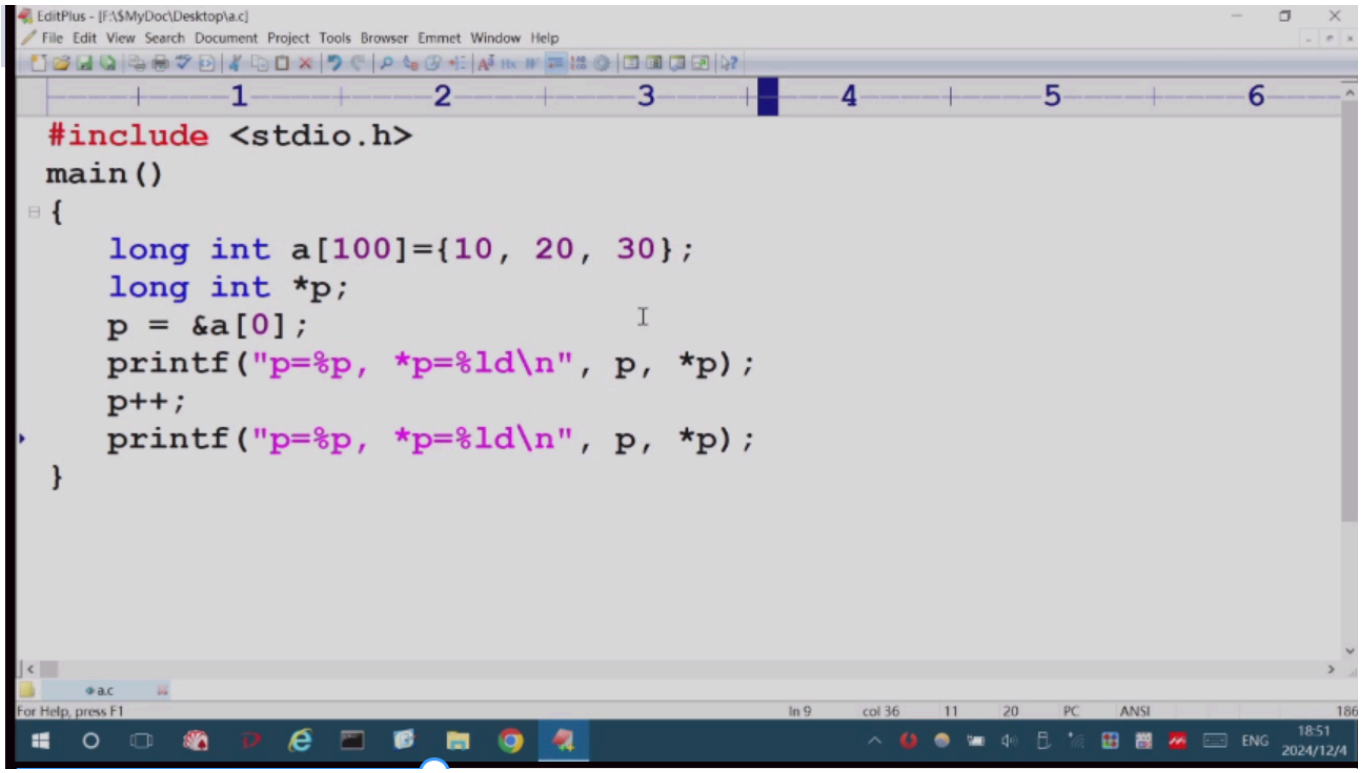
[[Pasted image 20241201120334.png

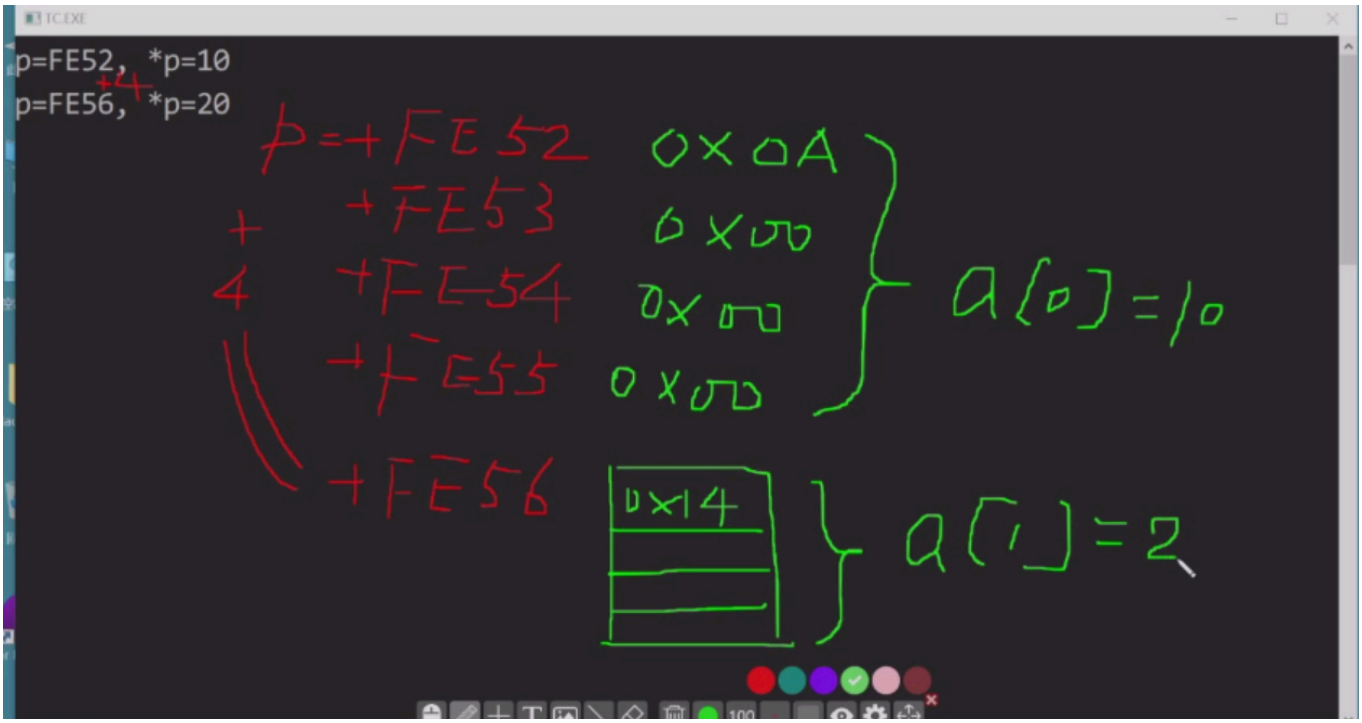


指针

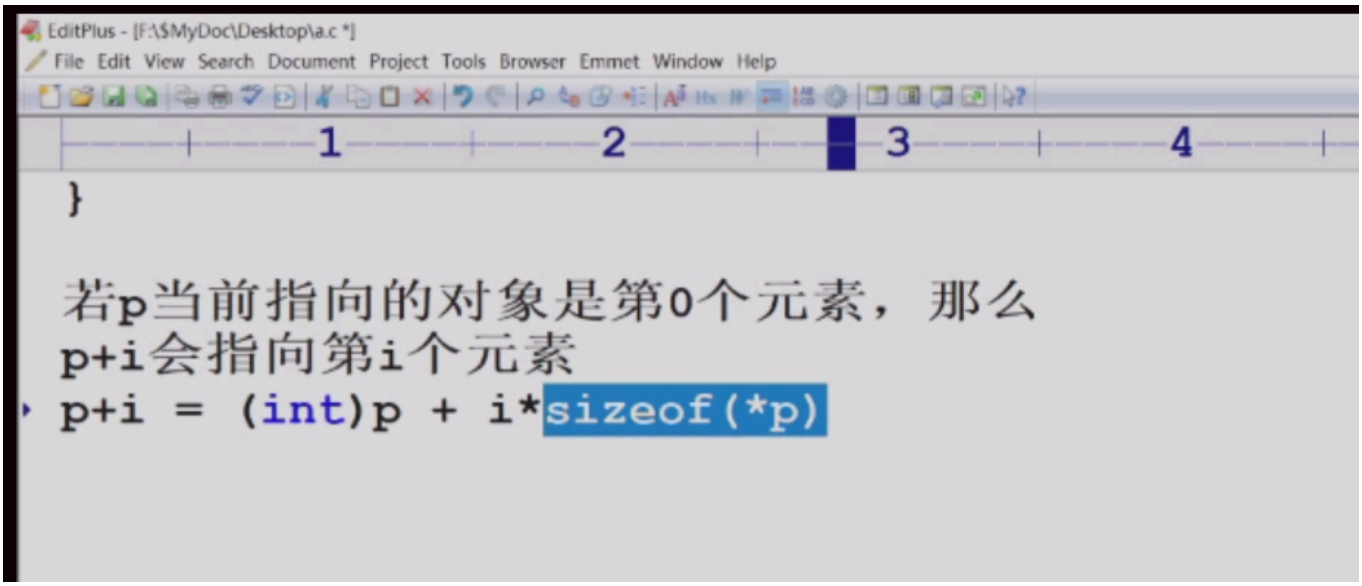
指针加减一个整数


EditPlus - [F:\$MyDoc\Desktop\a.c]
File Edit View Search Document Project Tools Browser Emmet Window Help
#include <stdio.h>
main()
{
long int a[100]={10, 20, 30};
long int *p;
p = &a[0];
printf("p=%p, *p=%ld\n", p, *p);
p++;
printf("p=%p, *p=%ld\n", p, *p);
}
For Help, press F1


EditPlus - [F:\$MyDoc\Desktop\a.c]
File Edit View Search Document Project Tools Browser Emmet Window Help
#include <stdio.h>
main()
{
long int a[100]={10, 20, 30};
long int *p;
p = &a[0];
printf("p=%p, *p=%ld\n", p, *p);
p++;
printf("p=%p, *p=%ld\n", p, *p);
}
For Help, press F1



p+i的公式



当 `*p` 的类型是char，那么p+i就是p+i

指针和指针的差

> **Important**
>
> 不等于地址之差

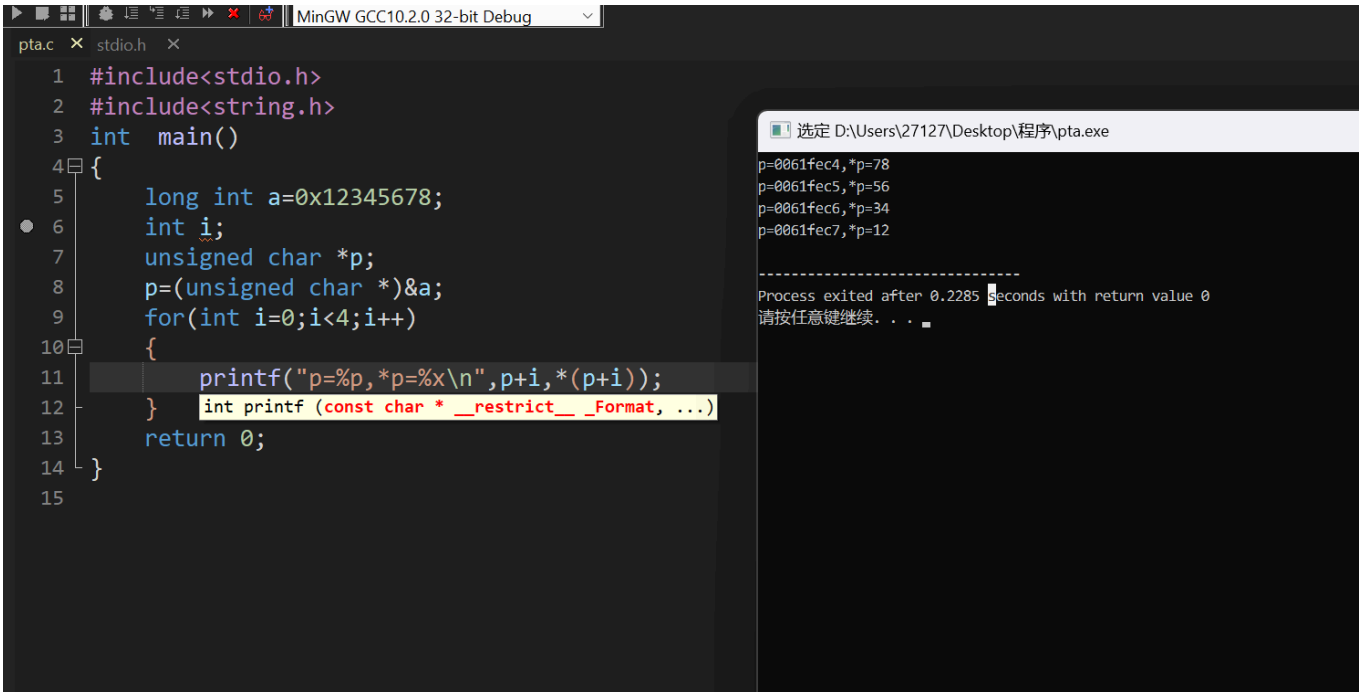
}

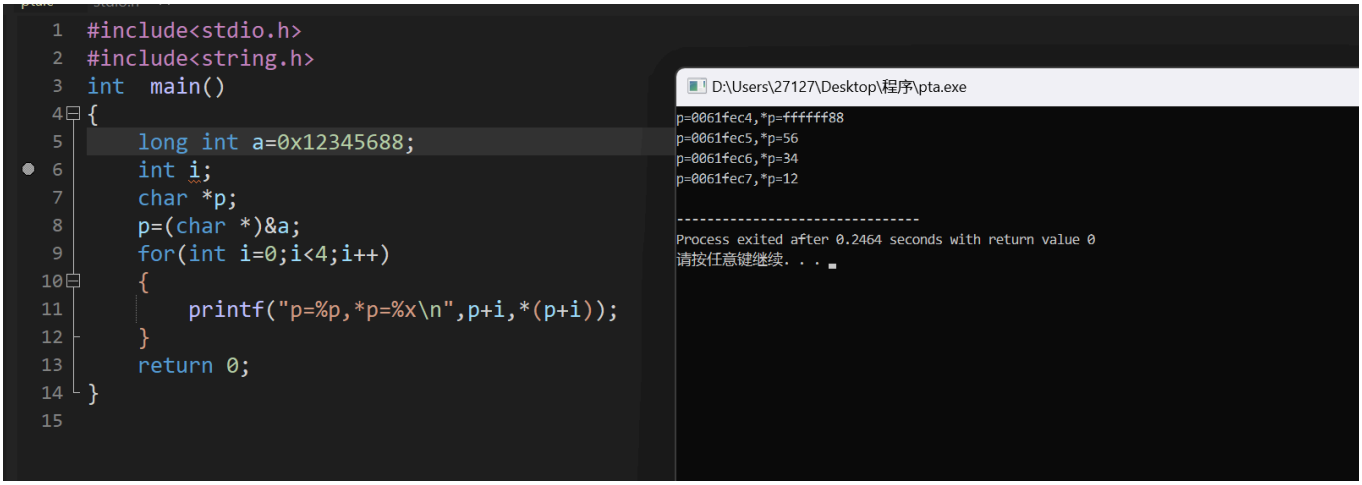
若p当前指向的对象是第0个元素，那么
p+i会指向第i个元素
p+i = (int)p + i*sizeof(*p)

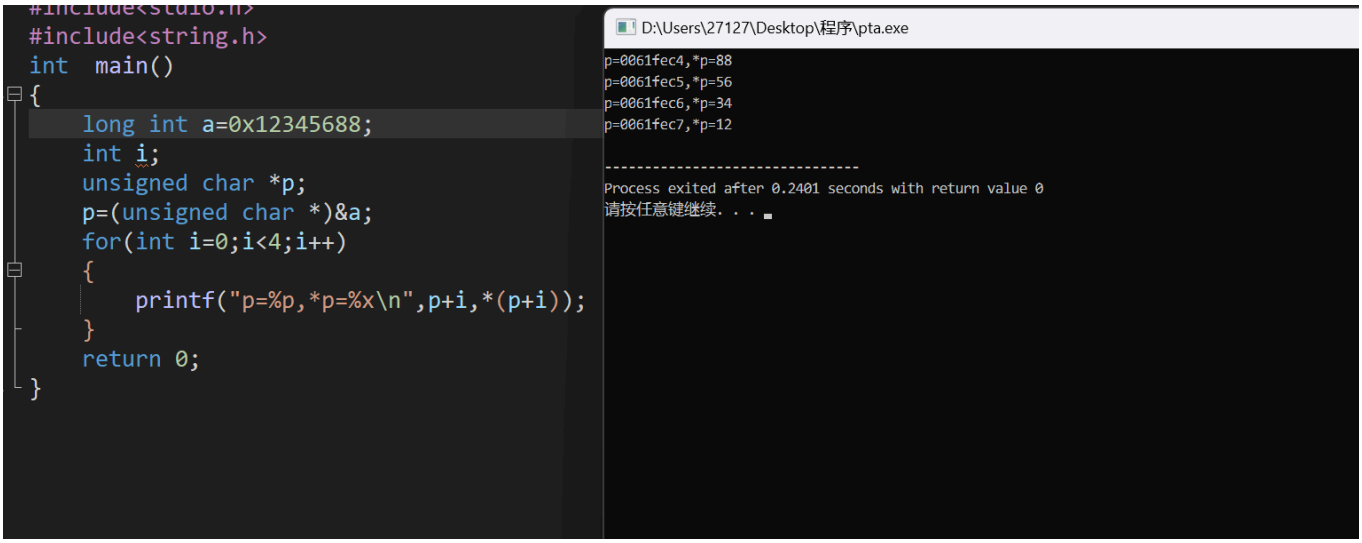
只有q、p同类型时，才可以做减法运算
q-p表示这两个指针所指向的对象之间相差的元素个数
q-p = ((int)q - (int)p) / sizeof(*p)

指针作比较

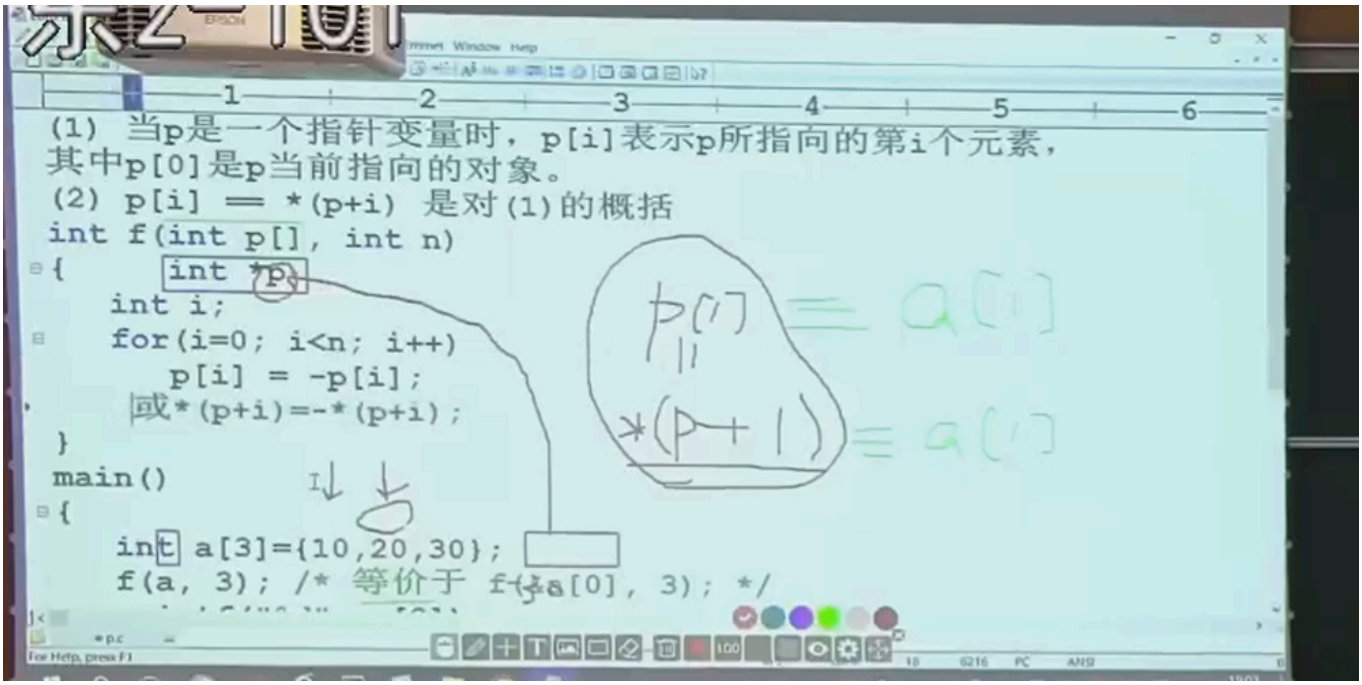
指针输出时的扩充问题

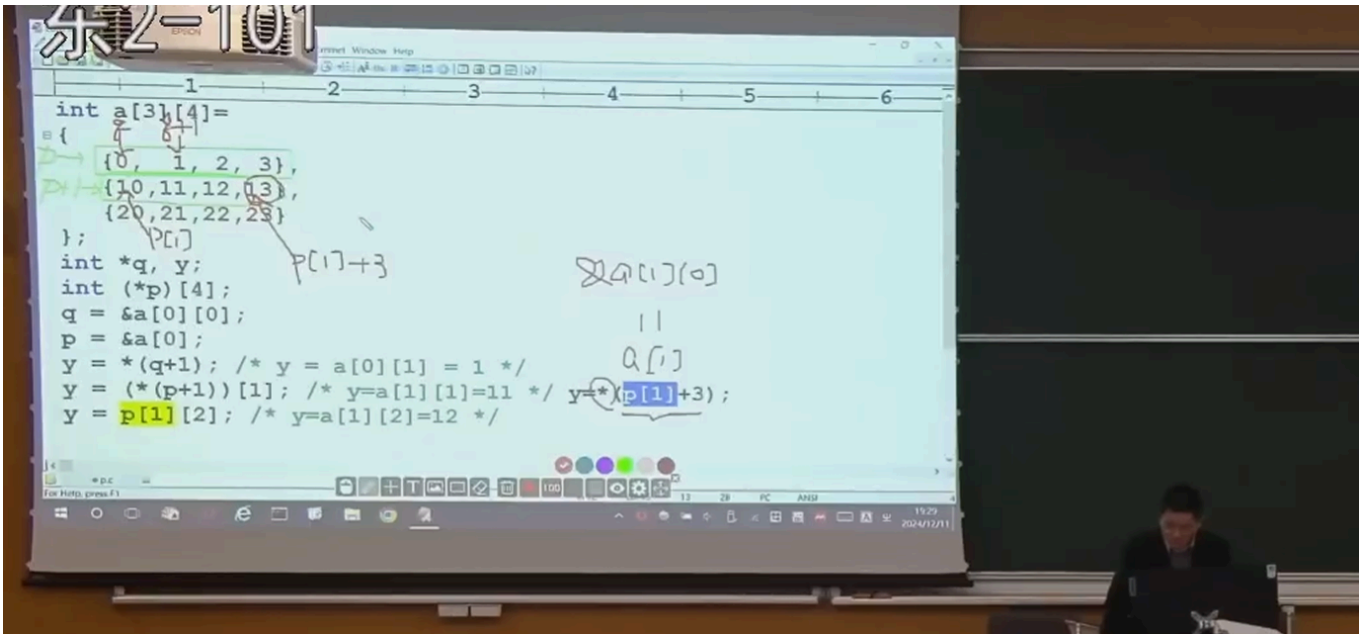


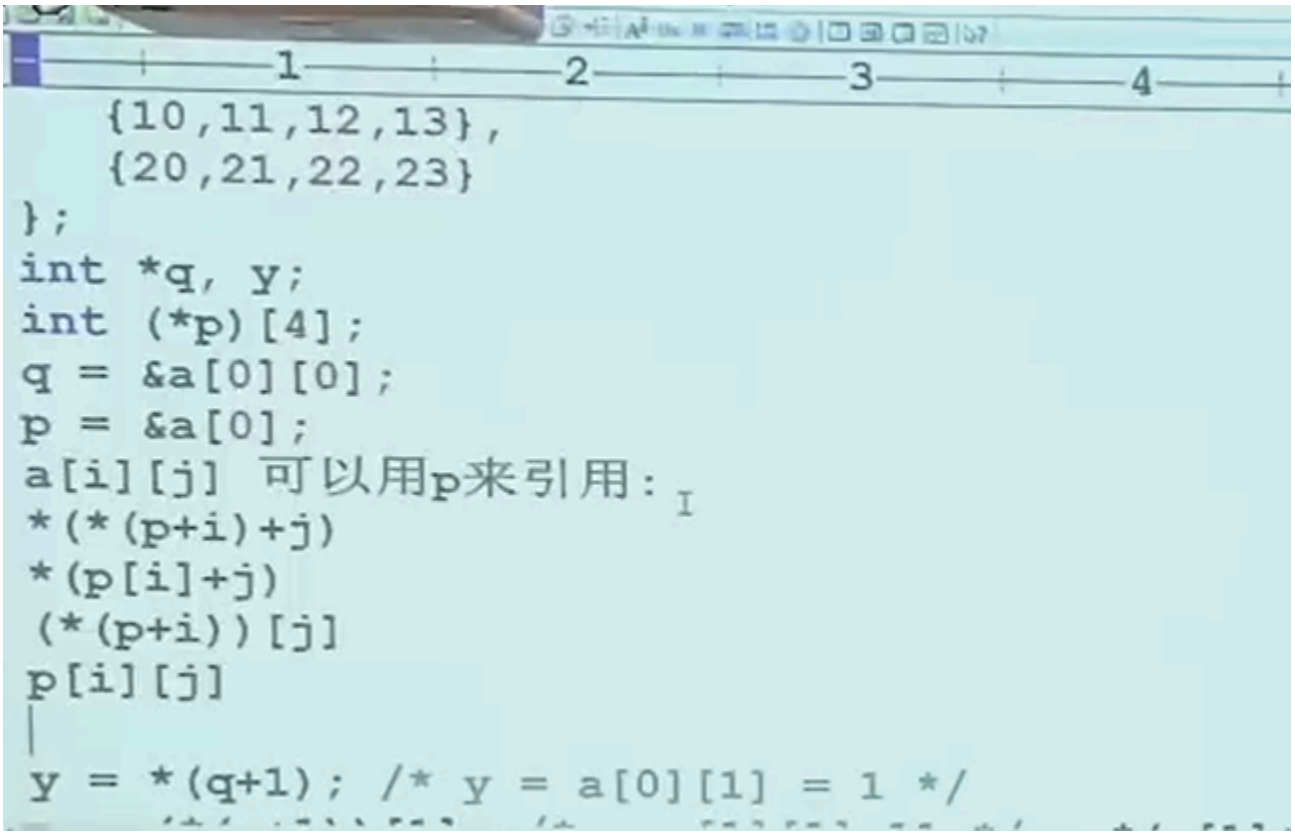



#include<stdio.h>
#include<string.h>
int main()
{
long int a=0x12345688;
int i;
unsigned char *p;
p=(unsigned char *)&a;
for(int i=0;i<4;i++)
{
printf("p=%p,*p=%x\n",p+i,*(p+i));
}
return 0;
}
D:\Users\27127\Desktop\程序\pta.exe
p=0061fec4,*p=88
p=0061fec5,*p=56
p=0061fec6,*p=34
p=0061fec7,*p=12
Process exited after 0.2401 seconds with return value 0
请按任意键继续. . .

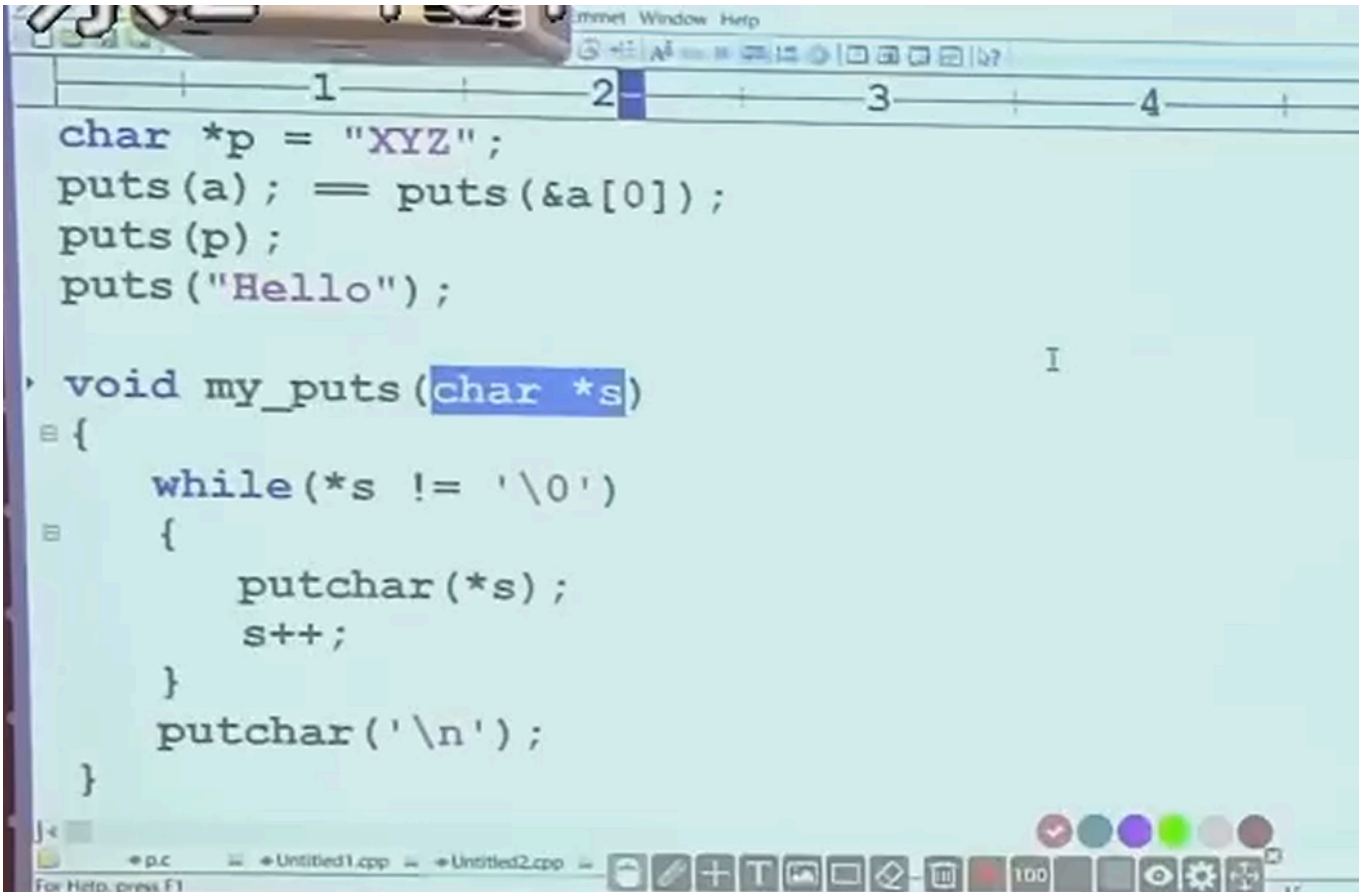
指针和数组


(1) 当p是一个指针变量时，p[i]表示p所指向的第i个元素，
其中p[0]是p当前指向的对象。
(2) p[i] == *(p+i) 是对(1)的概括
int f(int p[], int n)
{
int *p;
int i;
for(i=0; i<n; i++)
p[i] = -p[i];
或*(p+i)=-*(p+i);
}
main()
{
int a[3]={10,20,30};
f(a, 3); /* 等价于 f(&a[0], 3); */

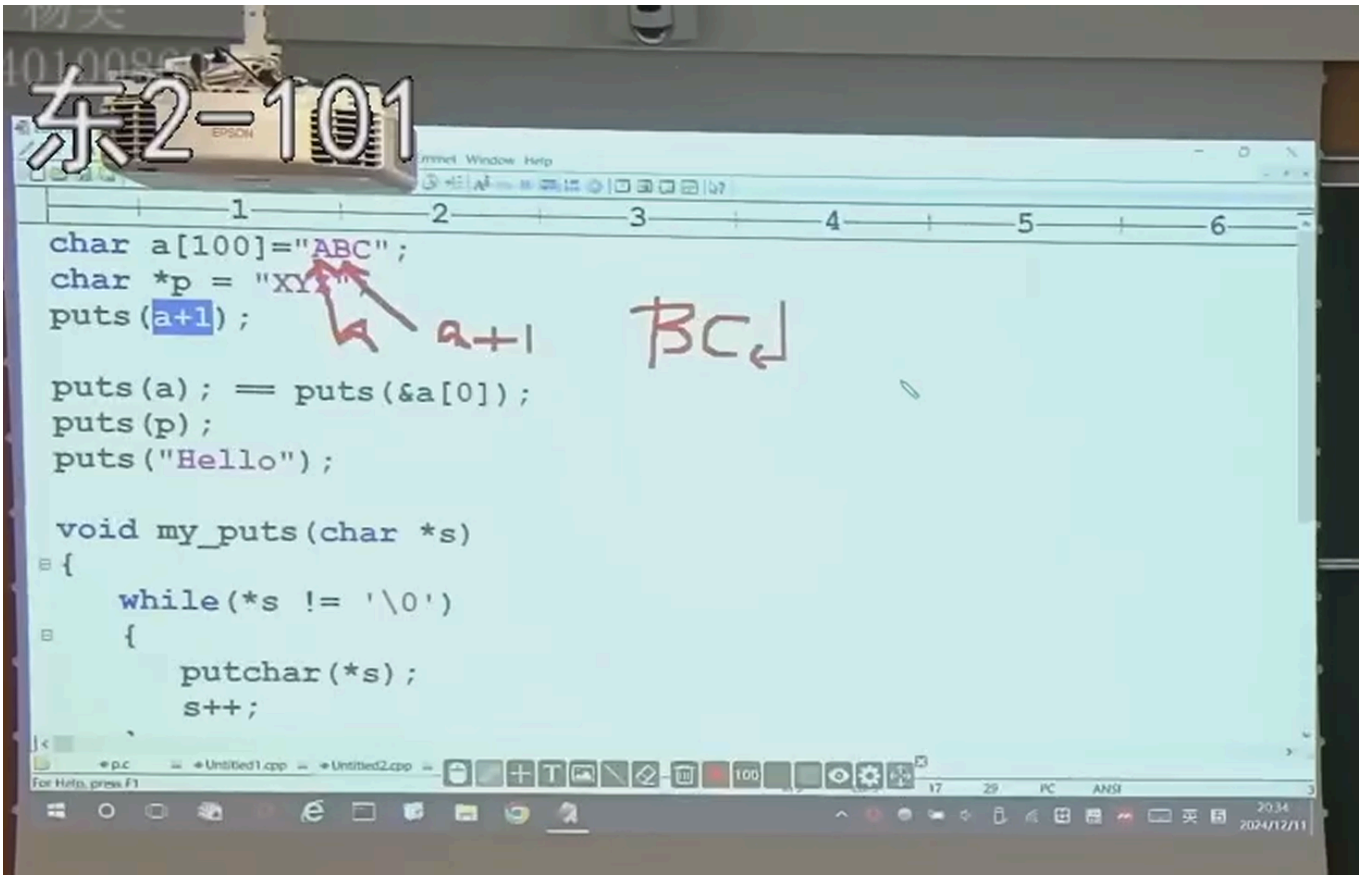

int a[3][4]=
{
{0, 1, 2, 3},
{10,11,12,13},
{20,21,22,23}
};
int *q, y;
int (*p)[4];
q = &a[0][0];
p = &a[0];
y = *(q+1); /* y = a[0][1] = 1 */
y = (*(p+1))[1]; /* y=a[1][1]=11 */
y = p[1][2]; /* y=a[1][2]=12 */



```
    {10,11,12,13},
    {20,21,22,23}
};
int *q, y;
int (*p)[4];
q = &a[0][0];
p = &a[0];
a[i][j] 可以用p来引用:
*(*(p+i)+j)
*(p[i]+j)
(*(p+i))[j]
p[i][j]

y = *(q+1); /* y = a[0][1] = 1 */
```



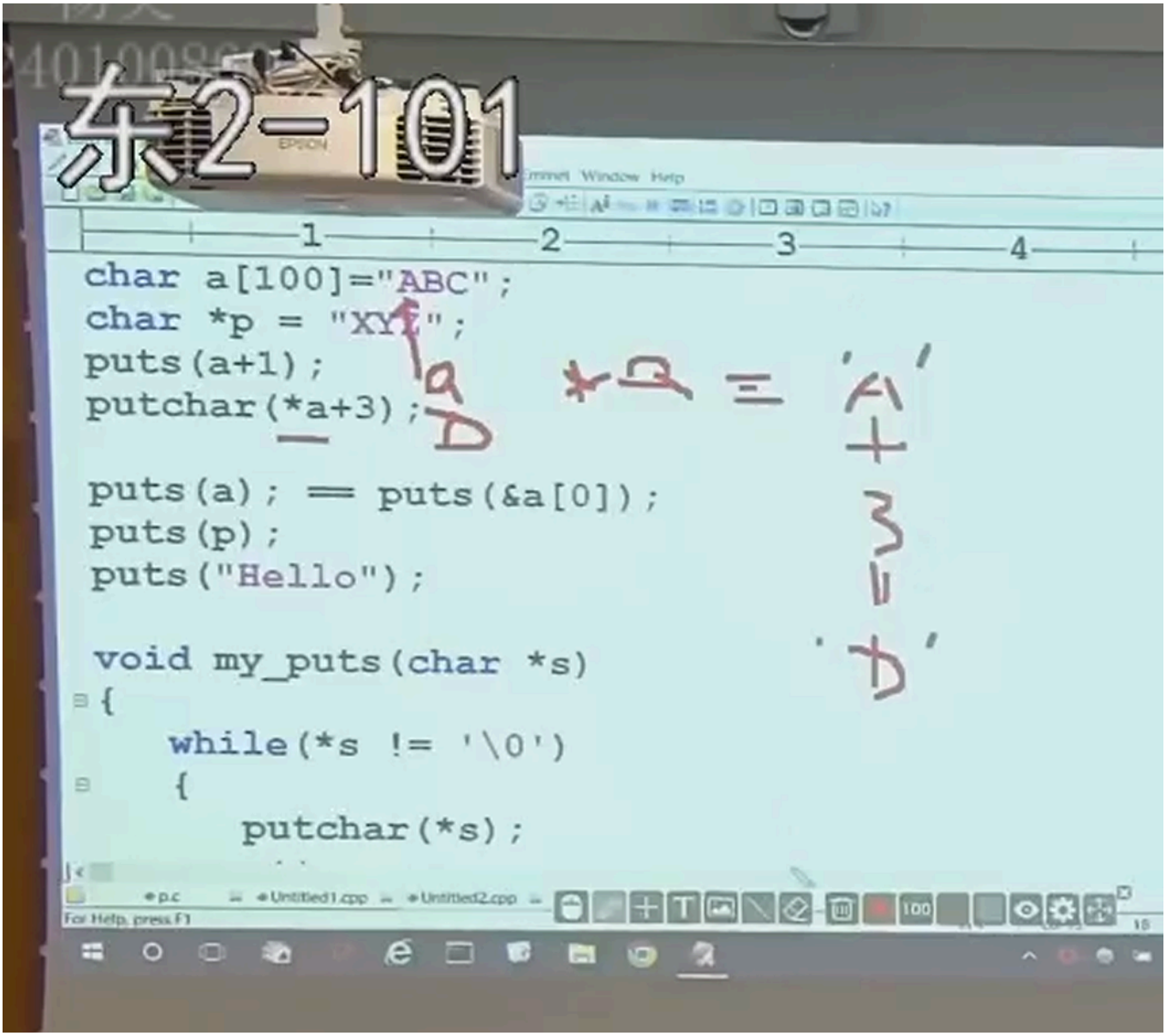
```
char *p = "XYZ";
puts(a); == puts(&a[0]);
puts(p);
puts("Hello");

void my_puts(char *s)
{
    while(*s != '\0')
    {
        putchar(*s);
        s++;
    }
    putchar('\n');
}
```

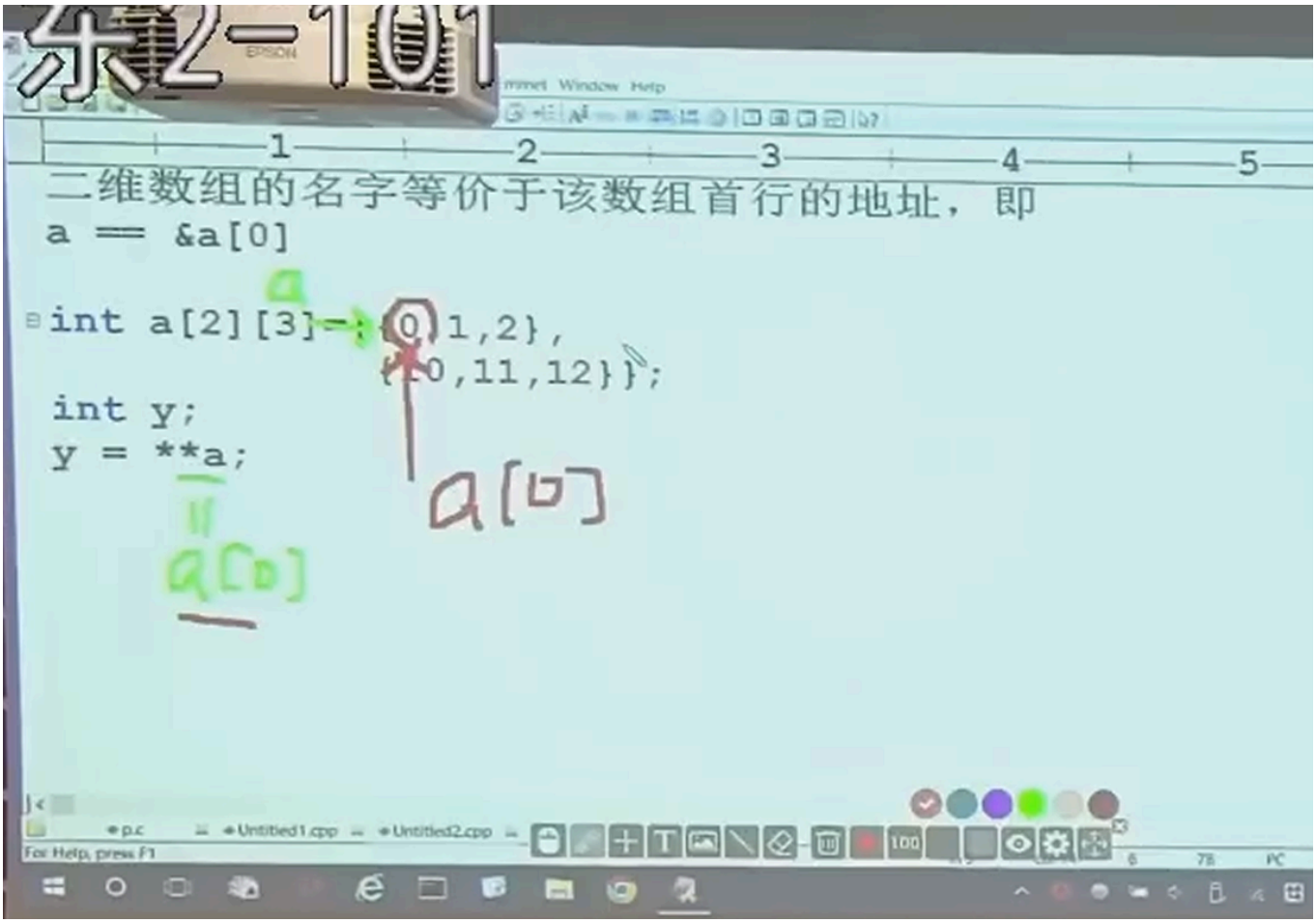


puts（a+1)会输出BC\n*

> **Note**
>
> 因为puts里面是地址

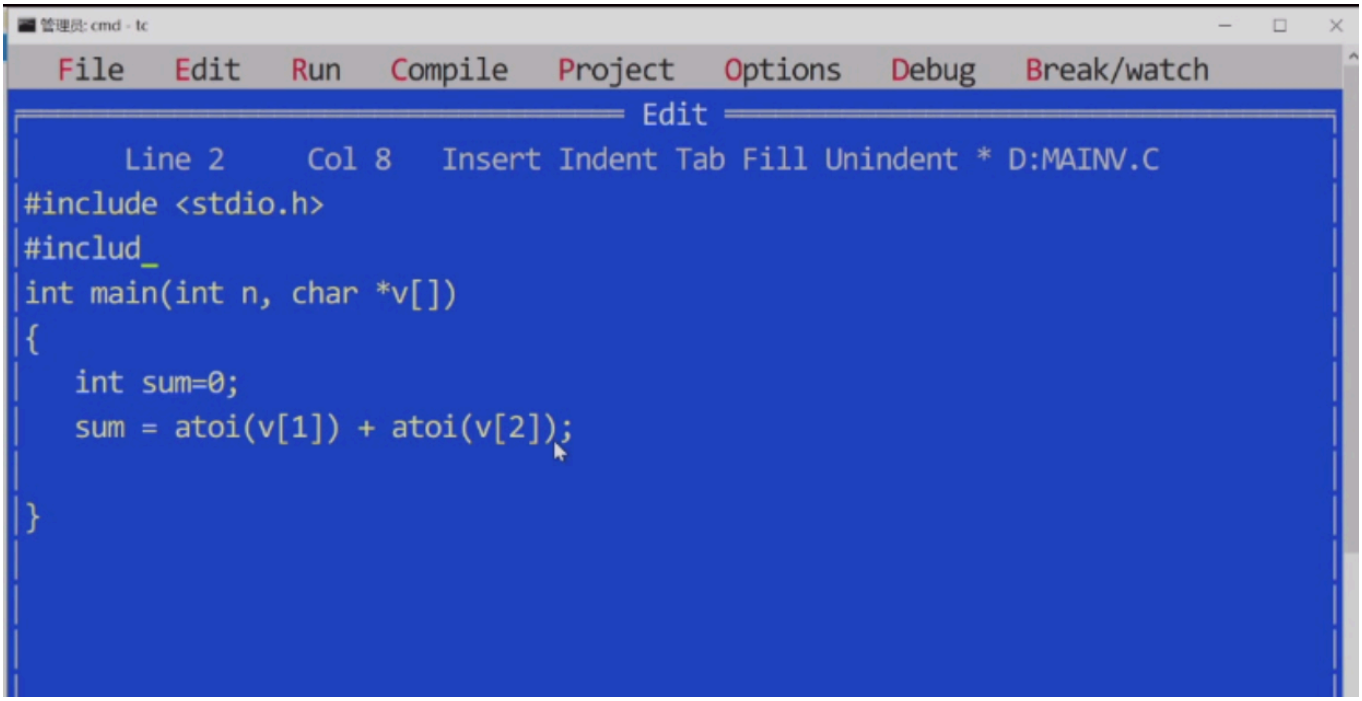
putchar里面不是地址，putchar（`*a`+3）输出D



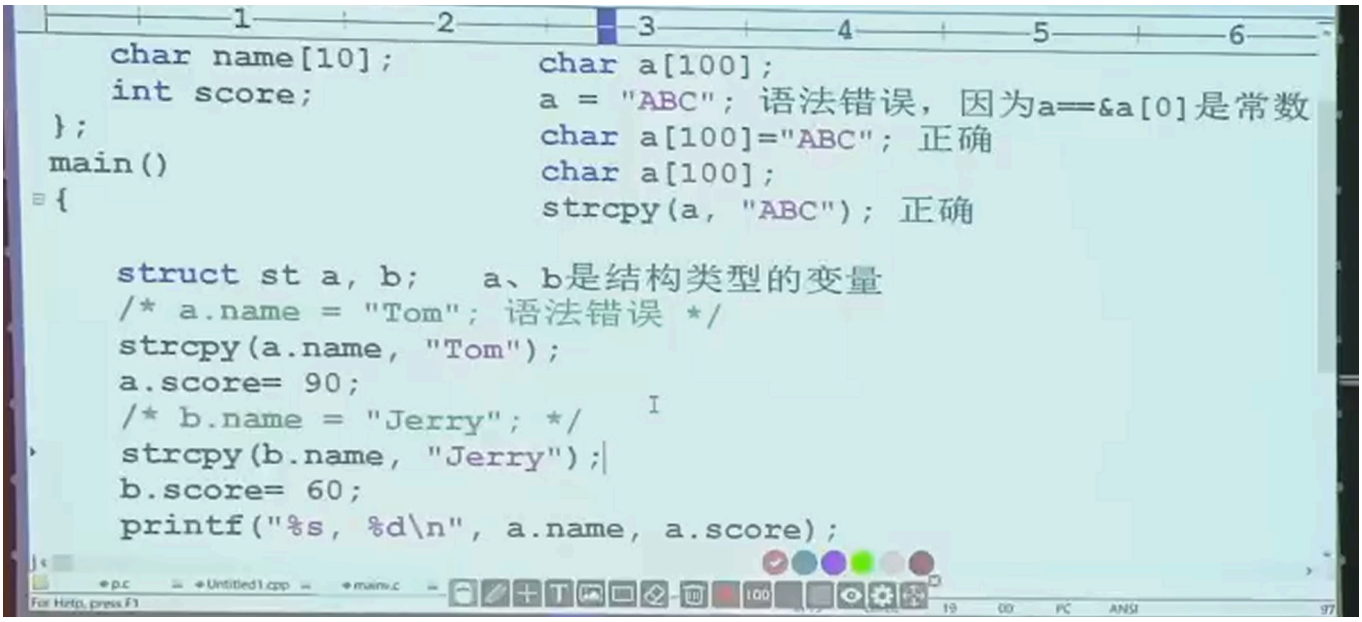


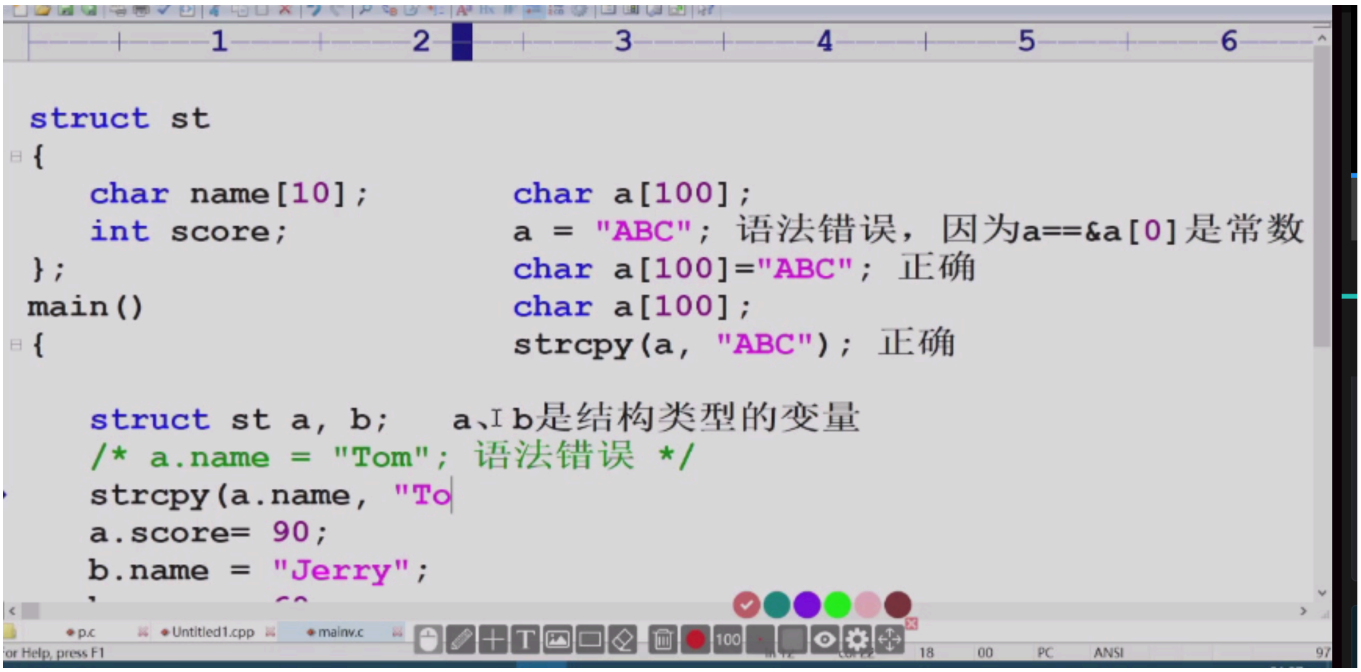
`a`就是a【0】【0】**

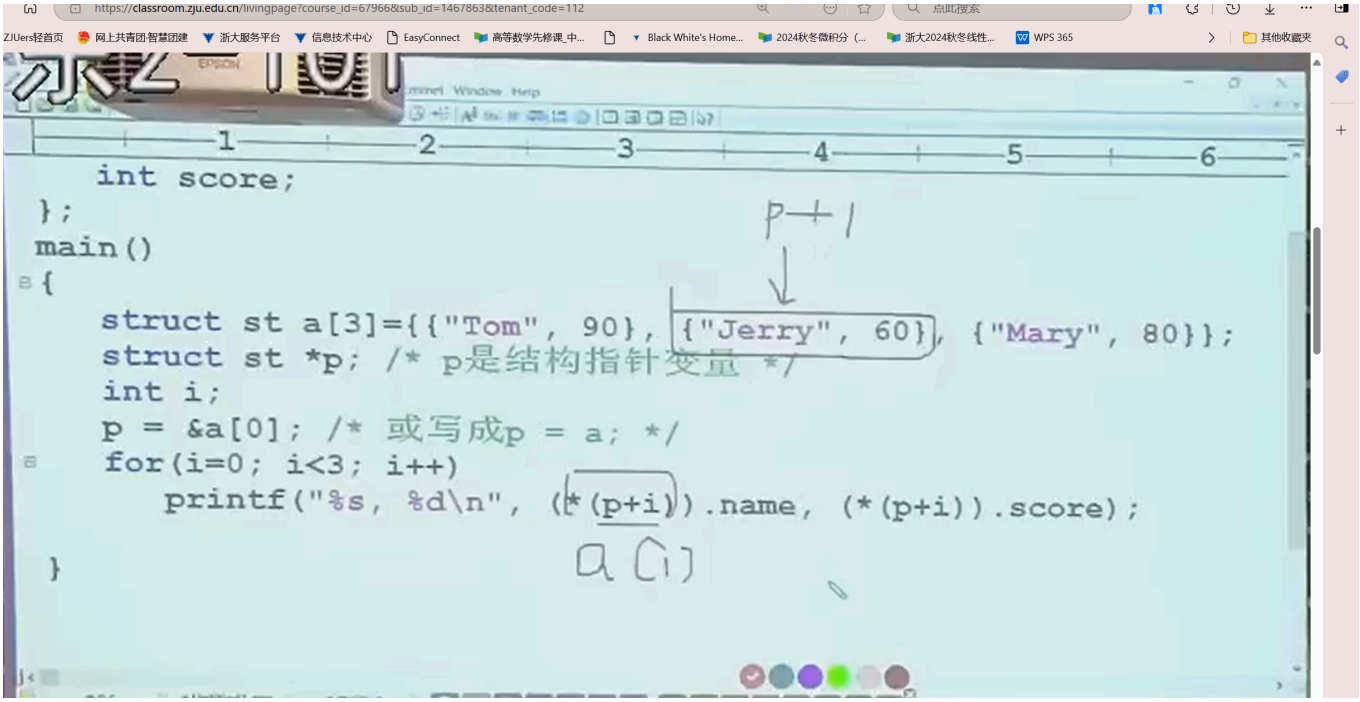
命令行参数

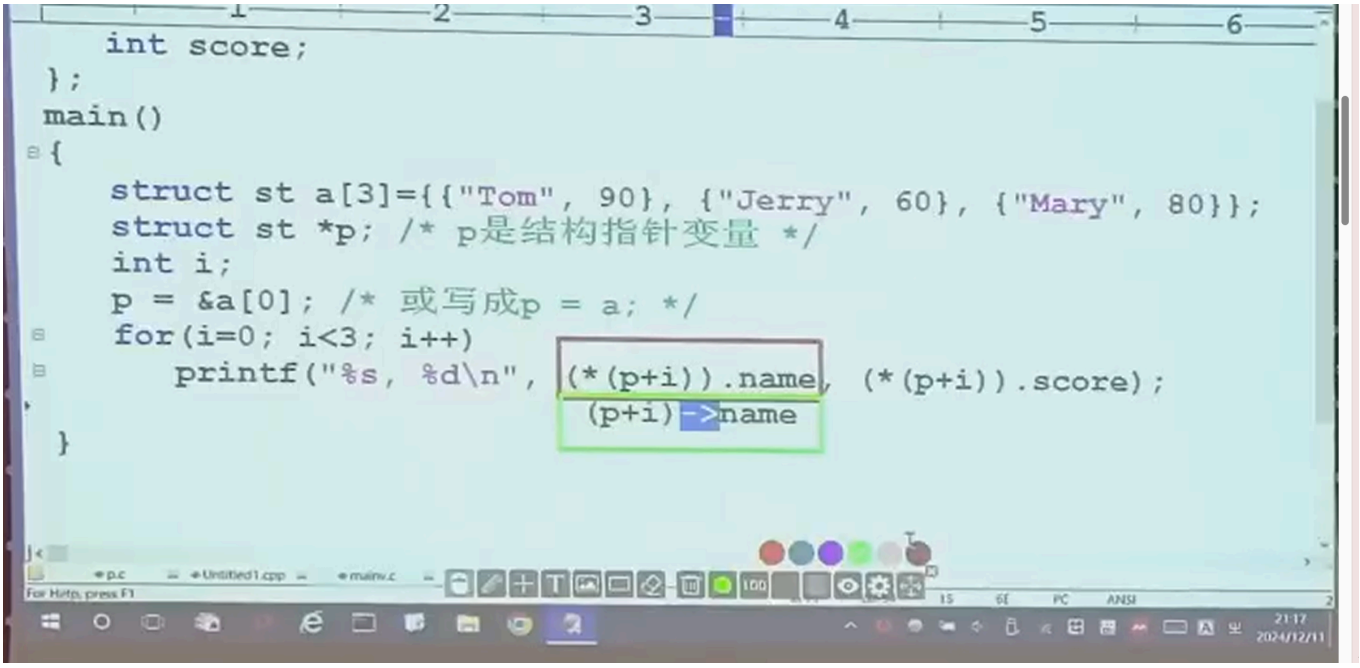


结构体

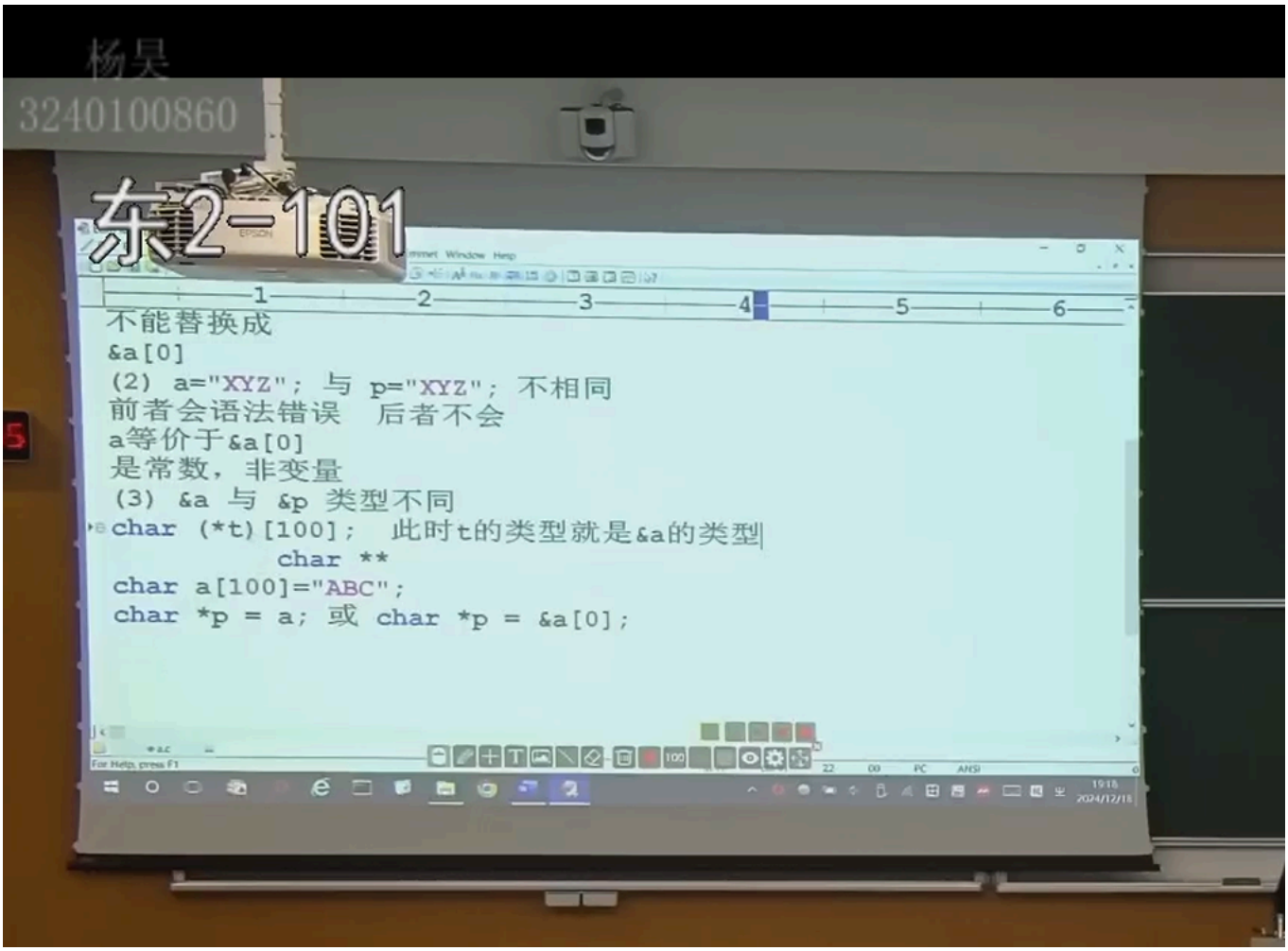

char name[10];
int score;
};
main()
{
struct st a, b; a、b是结构类型的变量
/* a.name = "Tom"; 语法错误 */
strcpy(a.name, "Tom");
a.score= 90;
/* b.name = "Jerry"; */
strcpy(b.name, "Jerry");
b.score= 60;
printf("%s, %d\n", a.name, a.score);
char a[100];
a = "ABC"; 语法错误，因为a==&a[0]是常数
char a[100]="ABC"; 正确
char a[100];
strcpy(a, "ABC"); 正确


struct st
{
char name[10];
int score;
};
main()
{
struct st a, b; a、b是结构类型的变量
/* a.name = "Tom"; 语法错误 */
strcpy(a.name, "To
a.score= 90;
b.name = "Jerry";
char a[100];
a = "ABC"; 语法错误，因为a==&a[0]是常数
char a[100]="ABC"; 正确
char a[100];
strcpy(a, "ABC"); 正确


int score;
};
main()
{
struct st a[3]={{"Tom", 90}, {"Jerry", 60}, {"Mary", 80}};
struct st *p; /* p是结构指针变量 */
int i;
p = &a[0]; /* 或写成p = a; */
for(i=0; i<3; i++)
printf("%s, %d\n", (*(p+i)).name, (*(p+i)).score);
}
p+1
a[1]


int score;
};
main()
{
struct st a[3]={{"Tom", 90}, {"Jerry", 60}, {"Mary", 80}};
struct st *p; /* p是结构指针变量 */
int i;
p = &a[0]; /* 或写成p = a; */
for(i=0; i<3; i++)
printf("%s, %d\n", (*(p+i)).name, (*(p+i)).score);
(p+i)->name
}

```
malloc()
typedef
数组与指针的区别
char a[100]="ABC";
char *p = a; 或 char *p = &a[0];
p[1] == a[1]
*(p+1) == *(a+1)
(1) sizeof(a)=100 与 sizeof(p)=4 不相同
此处的a
不能替换成
&a[0]
(2) a="XYZ"; 与 p="XYZ"; 不相同
前者会语法错误 后者不会
```



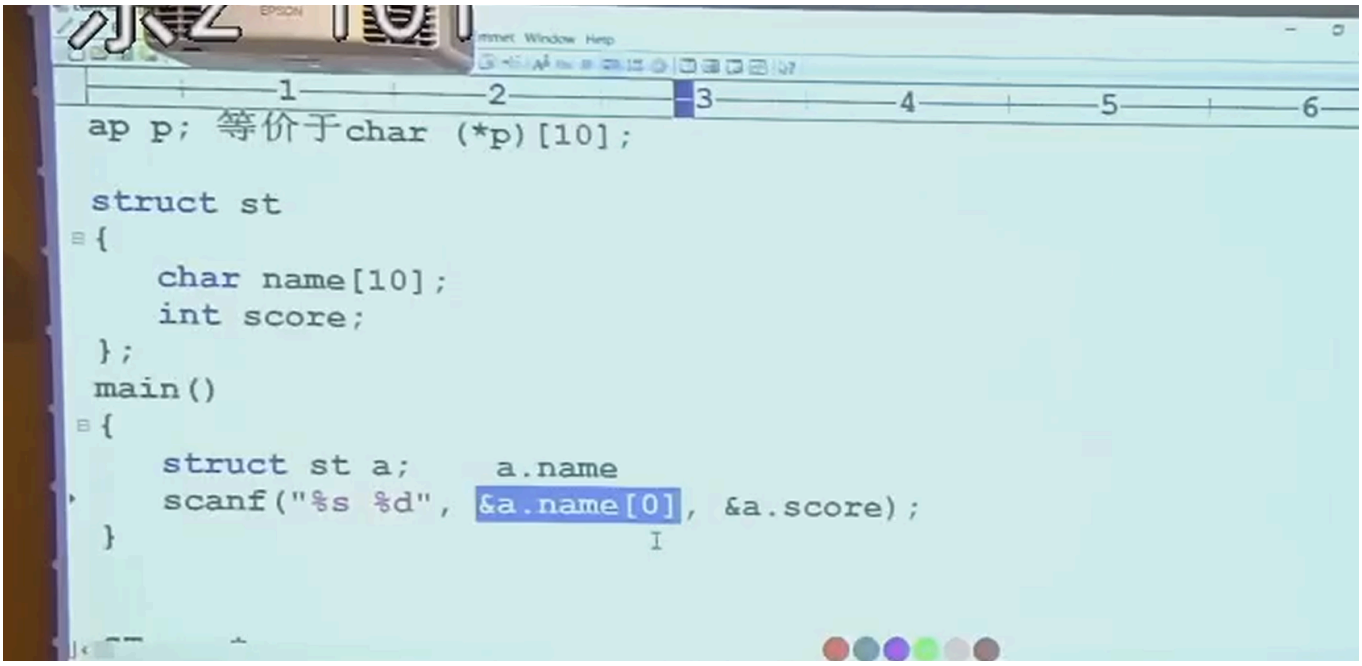
```
malloc()
typedef

typedef unsigned long int dword;
把unsigned long int这个类型替换成dword这个词
dword a; 相当于unsigned long int a;
typedef int tenint[10];

tenint x; 相当于int x[10];

typedef int tenint[10];
tenint x[2]; 相当于int x[2][10];
```

```
typedef int tenint[10];
tenint x[2]; 相当于int x[2][10];
定义一个数组指针类型ap，它指向一个由10个元素构成的数组，
数组的每个元素均为char
typedef char (*ap)[10];
ap p; 等价于char (*p)[10];
```

```
#include <stdlib.h>
main()
{
    char *p;
    p = malloc(100); /* malloc()返回内存块的首地址设为1000 */
    gets(p); /* 假设输入ABC并回车 */
    puts(p);
    free(p); /* 释放这块内存 */

}
```

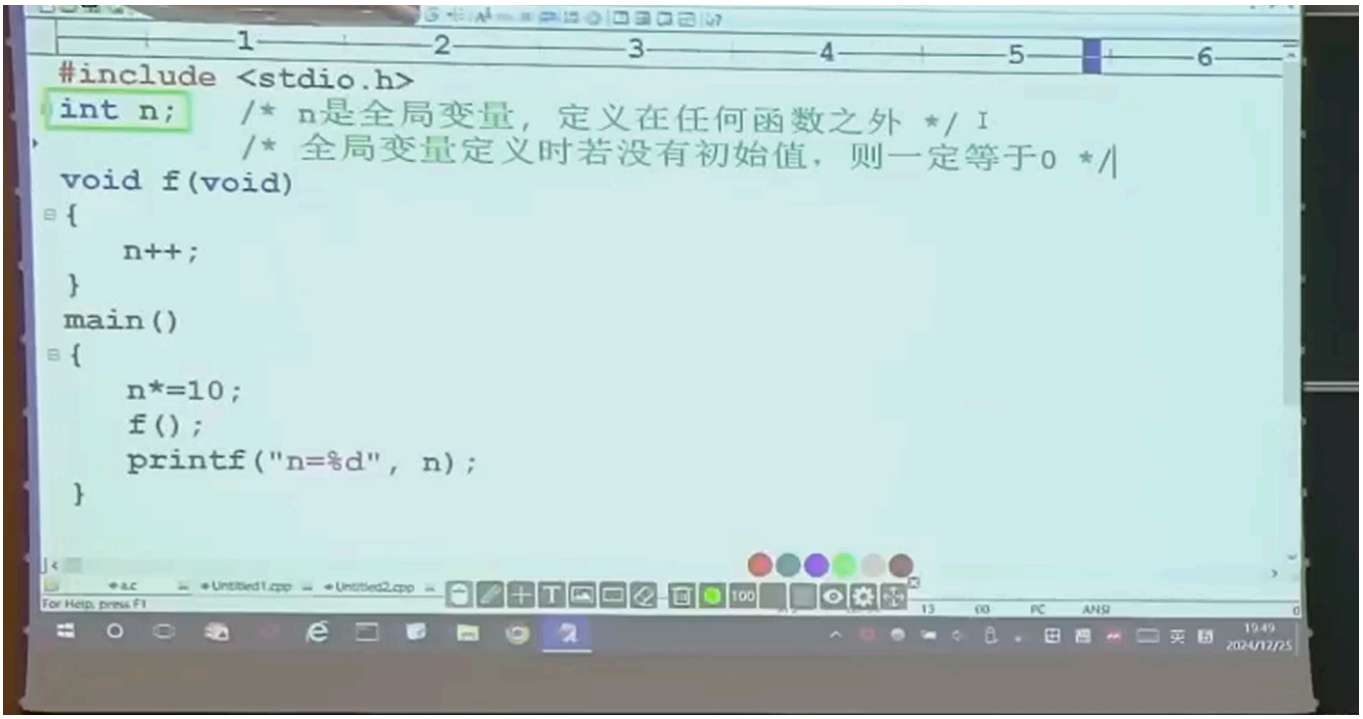
```
typedef char (*ap)[10];
ap p; 等价于char (*p)[10];

struct st
{
    char *name;
    int score;
} a, b, c[3];
```

```
typedef char (*ap)[10];
ap p; 等价于char (*p)[10];

typedef struct st
{
    char *name;
    int score;
} ST;

ST a, *p;
```



> 注意!
>
> **&a是不对的，但是程序跑出来是对的**

变量



```c
#include <stdio.h>
int n;    /* n是全局变量，定义在任何函数之外 */
          /* 全局变量定义时若没有初始值，则一定等于0 */
void f(void)
{
    n++;
}
main()
{
    n*=10;
    f();
    printf("n=%d", n);
}
```

定义该变量的语句所在的代码块即用{}括起来的复合语句

作用域：指变量的有效范围，即变量可以在该区域内被引用

生命期：指变量从诞生到死亡的周期

2. 局部静态(local static)变量

(1) 定义在函数体内

(2) 局部静态的定义语句早在main()运行时就已经执行过，并且该变量也在那时诞生，故每次该变量所在的函数被调用时，定义该变量的语句并不被执行

(3) 局部静态变量并不会因为所在函数的调用结束而死亡

(4) 局部静态变量在定义时若没有初始值，则它的值一定等于0

定义该变量的语句后续不会再被执行！

局部变量在所在{}结束后就是死的

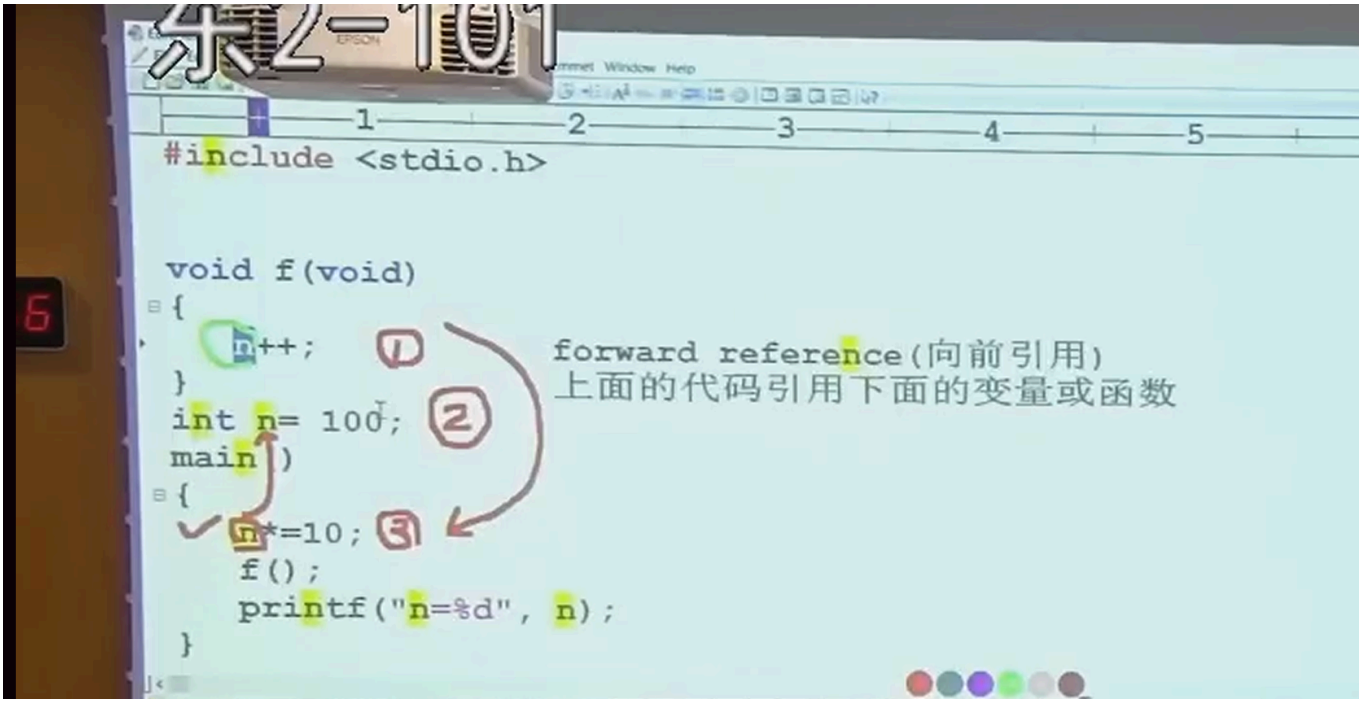
全局变量和局部静态变量一样，与main同生死

```c
#include <stdio.h>
int n= 100;    /* n是全局变量，定义在任何函数之外 */
          /* 全局变量定义时若没有初始值，则一定等于0 */
          /* 全局变量跟main同生死 */
void f(void)
{
   n++;
}
main()
{
   n*=10;
   f();
   printf("n=%d", n);
}
```

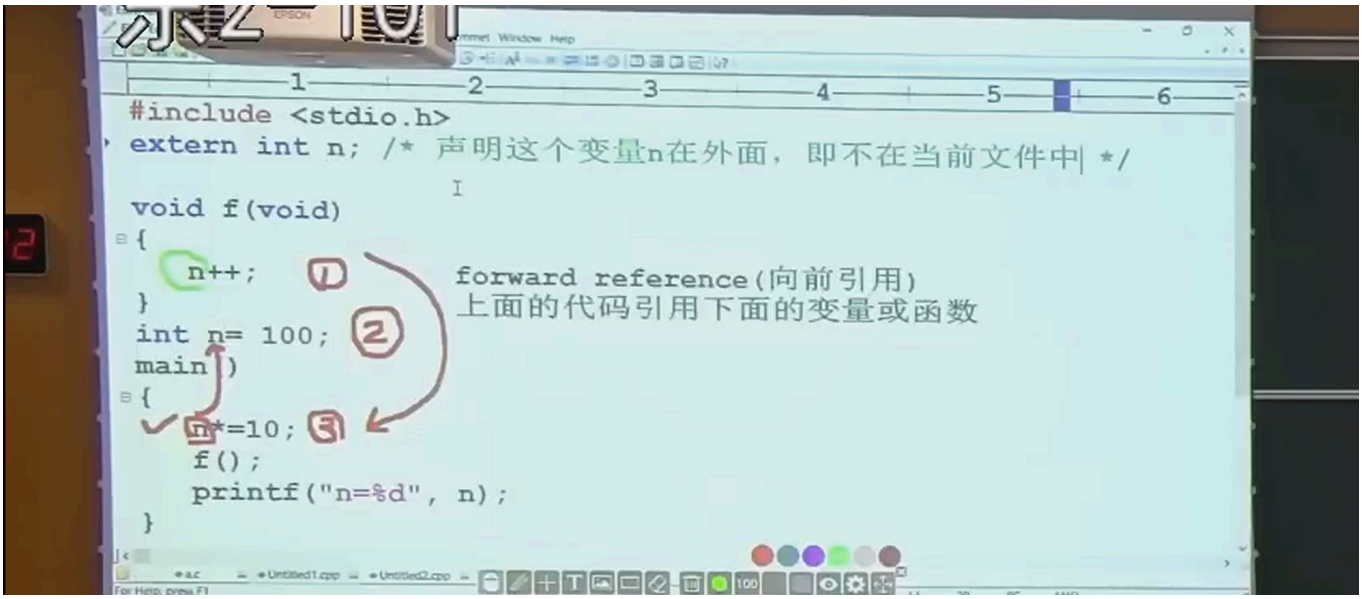
```c
#include <stdio.h>
static int n= 100;    /* n是全局静态变量，定义在任何函数之外 */
          /* 全局静态变量定义时若没有初始值，则一定等于0 */
          /* 全局静态变量跟main同生死 */
void f(void)
{
   int n = 5;
   n++;
}
main()
{
   n*=10;
   f();
   printf("n=%d", n);
}
```

```c
#include <stdio.h>
static int n=100;
void f(void){
    int n=0;
    n++;
}
int main(){
    n*=10;
    f();
    printf("%d",n);
    return 0;
}
```

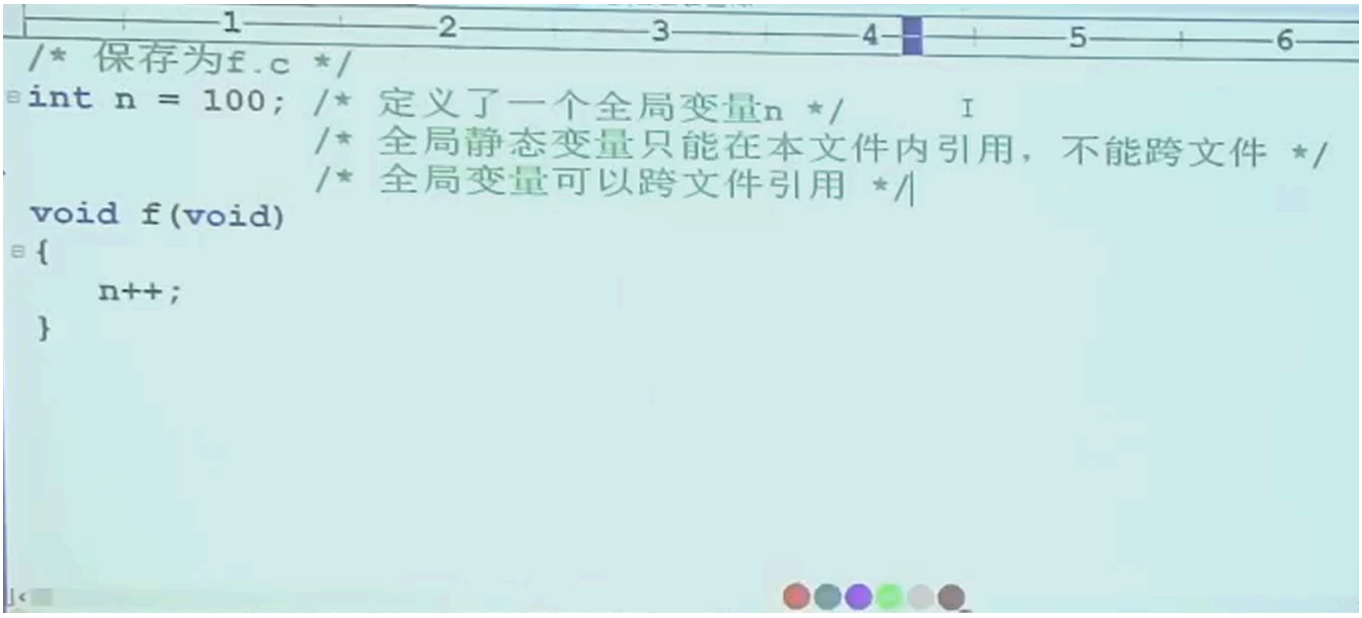
```
D:\Users\27127\Desktop\hh.exe
1000
--------------------------------
Process exited after 0.3019 seconds with return va
请按任意键继续. . .
```



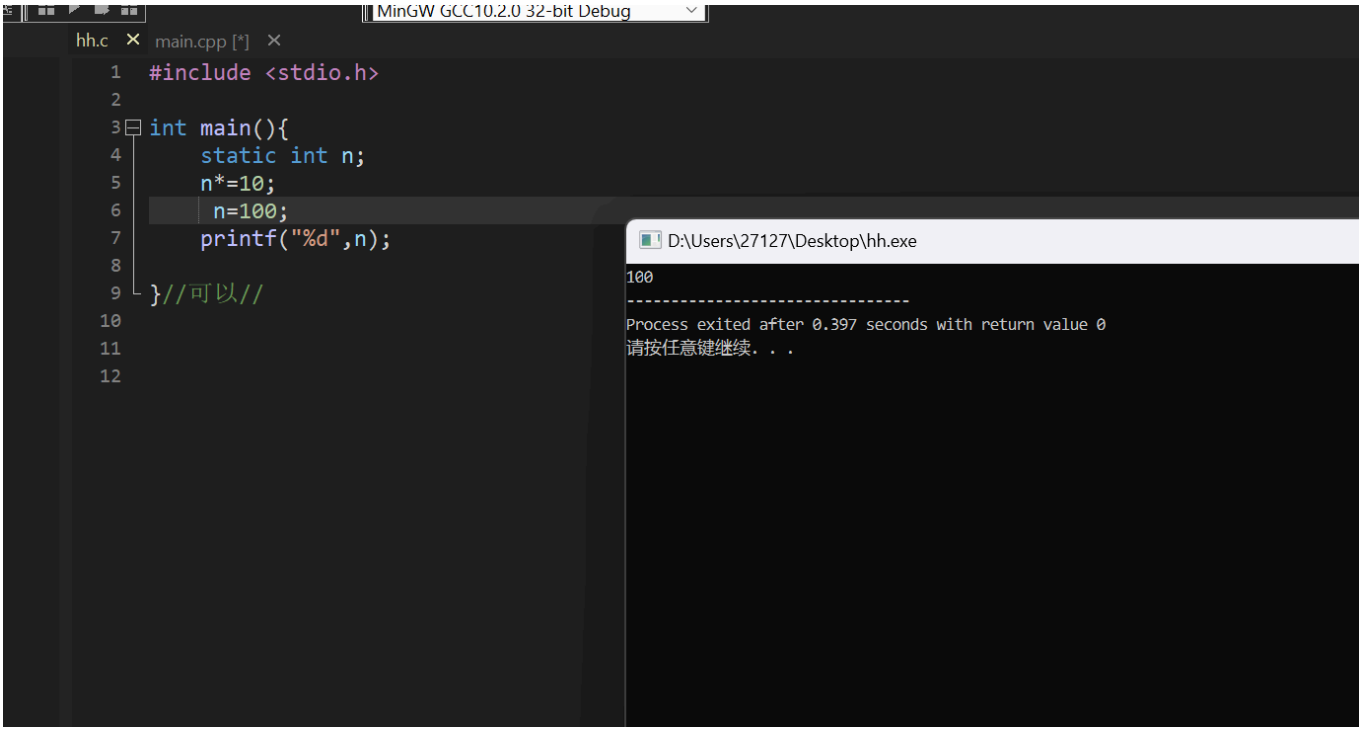
会报错!



欺骗编译器n在外面（其实在里面），这样就不报错了

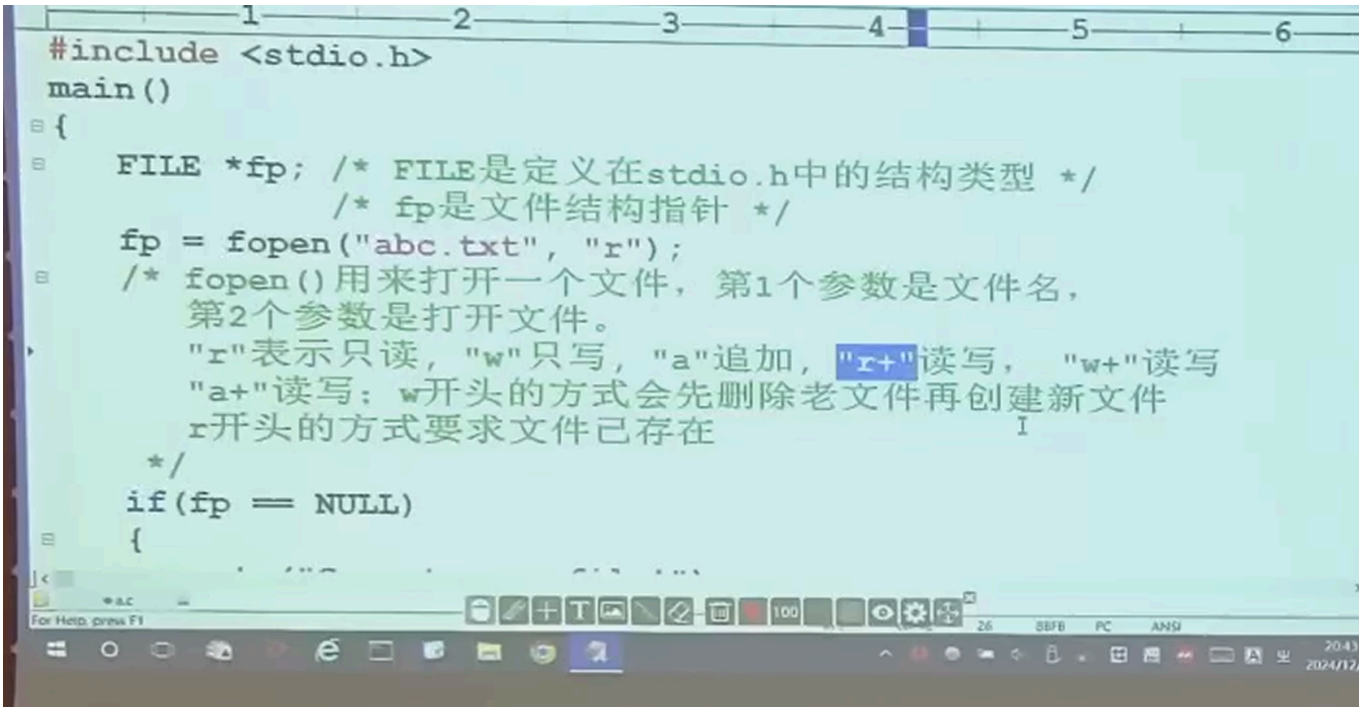


```c
#include <stdio.h>

int main(){
    n*=10;
    printf("%d",n);
    static int n=100;
    return 0;
}//不可以//
int main(){
    static int n;
    n*=10;
    printf("%d",n);
    n=100;
    return 0;
}//可以//
int main(){
    int n ;
    n*=10;
    printf("%d",n);
    n=100;
    return 0;
}//不可以//
```

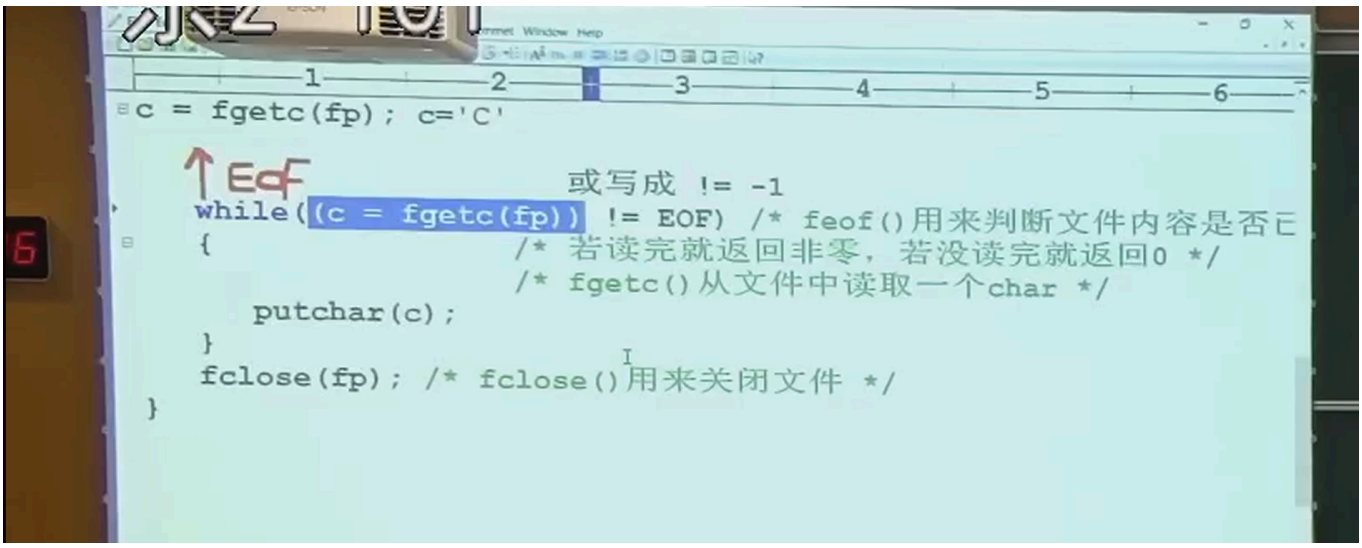

MinGW GCC10.2.0 32-bit Debug
hh.c
main.cpp [*]
#include <stdio.h>
int main(){
static int n;
n*=10;
n=100;
printf("%d",n);
}//可以//
D:\Users\27127\Desktop\hh.exe
100

Process exited after 0.397 seconds with return value 0
请按任意键继续. . .

文件



```c
#include <stdio.h>
main()
{
    FILE *fp; /* FILE是定义在stdio.h中的结构类型 */
              /* fp是文件结构指针 */
    fp = fopen("abc.txt", "r");
    /* fopen()用来打开一个文件，第1个参数是文件名，
       第2个参数是打开文件。
       "r"表示只读，"w"只写，"a"追加，"r+"读写，  "w+"读写
       "a+"读写；w开头的方式会先删除老文件再创建新文件
       r开头的方式要求文件已存在
     */
    if(fp == NULL)
    {
```



```c
c = fgetc(fp); c='C'

↑EOF                    或写成 != -1
    while((c = fgetc(fp)) != EOF) /* feof()用来判断文件内容是否已
    {                   /* 若读完就返回非零，若没读完就返回0 */
                        /* fgetc()从文件中读取一个char */
        putchar(c);
    }
    fclose(fp); /* fclose()用来关闭文件 */
}
```

```
    1          2          3          4
    puts("Cannot open file!");
    exit(0); /* 强制结束程序的运行 */
}
while(!feof(fp))
{
    fscanf(fp, "%d ", &x);
    sum += x;
}
fclose(fp); /* fclose()用来关闭文件 */
fclose(fp2);
```

```
#include <stdio.h>
main()
{
   fgets()从文件中读取一行内容
   fputs()写入一行字符串到文件中
   rewind(fp)把文件指针移到文件的最开头处
   fread(p, 1, n, fp); 把p指向的n个字节写入fp
   fwrite(p, 1, n, fp);
   int x, sum=0;
   FILE *fp, *fp2;
   fp = fopen("abc.txt", "r");
   fp2 = fopen("d:\\tc\\xyz.txt", "w");
   if(fp == NULL || fp2 == NULL)
   {
```


MinGW GCC10.2.0 32-bit Debug
hh.c
#include <stdio.h>
int main(){
FILE *fp,*fp2;
fp=fopen("abc.txt","r");
fp2=fopen("xyz.txt","w");
if(fp==NULL||fp2==NULL){
puts("wrong!");
exit(0);
}
while(!feof(fp)){
char c=fgetc(fp);
fputc(c,fp2);
}
fclose(fp);
fclose(fp2);
}
文件
编辑
ABC


#include <stdio.h>
int main(){
FILE *fp,*fp2;
fp=fopen("abc.txt","r");
fp2=fopen("xyz.txt","w");
if(fp==NULL||fp2==NULL){
puts("wrong!");
exit(0);
}
char c;
while((c=fgetc(fp))!=EOF){
fputc(c,fp2);
}
fclose(fp);
fclose(fp2);
}
文件
ABC