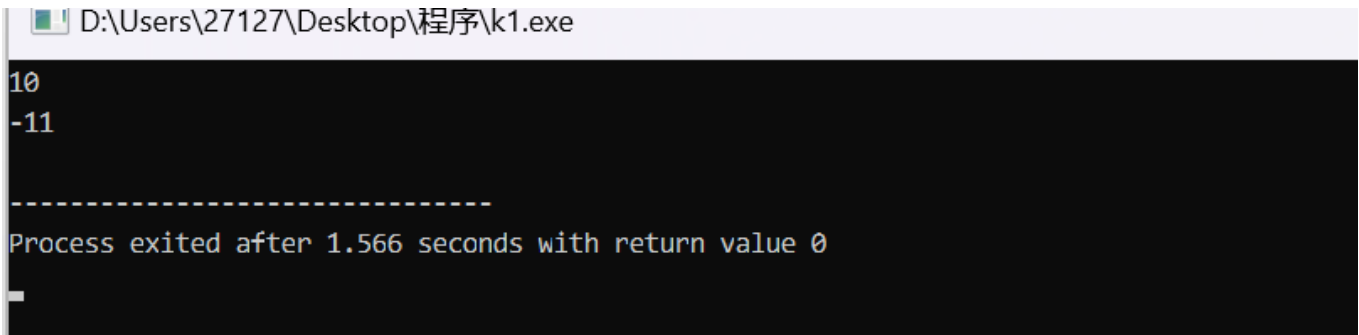


二进制求反

- 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

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

    return 0;
}
```



```
D:\Users\27127\Desktop\程序\k1.exe
10
-11
-----
Process exited after 1.566 seconds with return value 0
```

0000 1010

1111 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

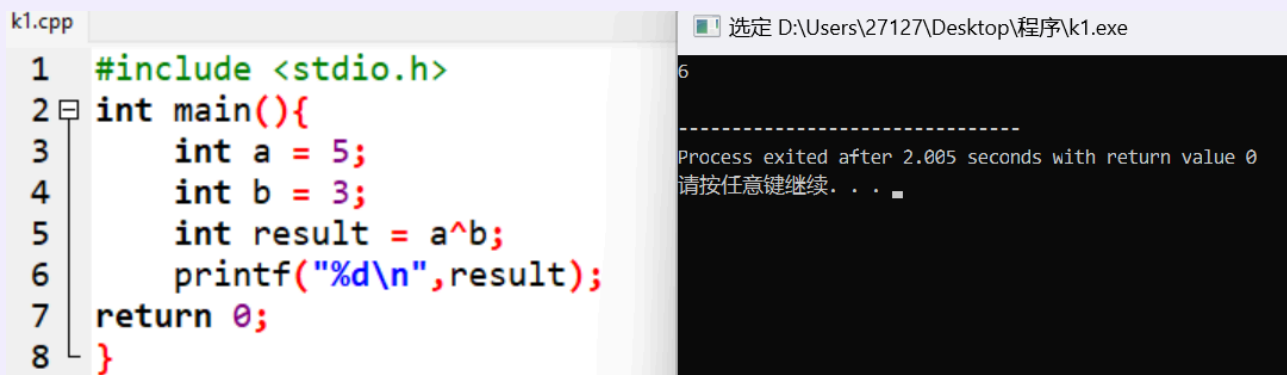
对两个操作数的对应位进行逐位比较，若两个比较的位不同，则结果为1；若相同则结果为0

🔥 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

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



```
k1.cpp
1 #include <stdio.h>
2 int main(){
3     int a = 5;
4     int b = 3;
5     int result = a^b;
6     printf("%d\n",result);
7     return 0;
8 }
```

选定 D:\Users\27127\Desktop\程序\k1.exe

```
6
-----
Process exited after 2.005 seconds with return value 0
请按任意键继续. . .
```

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

```
#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

```

#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

```
#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

```
C 11.c 2 X
D: > Users > 27127 > Desktop > c > C 11.c > main()
1  #include <stdio.h>
2
3  int main() {
4      printf("Size of long int: %zu bytes\n", sizeof(long int));
5      printf("Size of long long: %zu bytes\n", sizeof(long long));
6      printf("Size of char: %zu bytes\n", sizeof(char));
7      printf("Size of int: %zu bytes\n", sizeof(int));
8      printf("Size of short: %zu bytes\n", sizeof(short));
9      printf("Size of float: %zu bytes\n", sizeof(float));
10     printf("Size of double: %zu bytes\n", sizeof(double));
11     printf("Size of long double: %zu bytes\n", sizeof(long double));
12     return 0;
13 }
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
Size of long int: 4 bytes
Size of long long: 8 bytes
PS D:\Users\27127\Desktop> cd "d:\Users\27127\Desktop\c\" ; if ($?) { gcc 11.c -o 11 } ; if ($?) { .\11 }
Size of long int: 4 bytes
Size of long long: 8 bytes
Size of char: 1 bytes
Size of int: 4 bytes
Size of short: 2 bytes
Size of float: 4 bytes
Size of double: 8 bytes
Size of long double: 16 bytes
PS D:\Users\27127\Desktop\c> 
```

数组

```
#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越界。

```
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 $\&a[0][3]=1003$

越界

是否越界。

```
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 $\&a[0][3]=1003$

```
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
```

这里的a是形参 (formal parameter)

```
int my_strlen(char a[])
或 char a[100]
{
    int i=0;
    while(a[i] != '\0')
        i++;
    return i;
}
```

```
#include <stdio.h>
#include <string.h>
main()
{
    char a[100];
    int n;
    scanf("%s", &a[0]);
    n = my_strlen(a);
    这里a是用一维数组做实参
    (actual parameter/argument)
    a不能写成a[]或a[100]
    printf("n=%d\n", n);
}
```

用二维数组做形参

```
int f(int a[][3], int n, int m)
或 int a[2][3]
{
    int i, j, sum=0;
    for(i=0; i<n; i++)
    {
        for(j=0; j<m; j++)
            sum += a[i][j];
    }
    return sum;
}
```

```
main()
{
    int x[2][3]={{1,2,3}, {10,20,30}};
    int y;
    y = f(x, 2, 3);
    用二维数组做实参
}
```

绝对值

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

优先级

3.1 运算符和表达式

(1) 优先级 (precedence) 与结合性 (association)

在 TC 编辑状态下, 按两次 F1, 再选 “Precedence” 可以查看优先级与结合性表。

$y=3+4*5$; 相当于 $y=3+(4*5)$;

$y=(3+4)*5$;

$y=3*4/5$;

$y = -(x++)$;

left to right左结合

right to left右结合

```
C.cpp x stdio.h x
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     int a = 1;
6     int b=2*++a;
7     return 0;
8 }
```

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

x--; 与 --x; 等价 都表示 $x=x-1$

已知 $x=2, y=3, z;$

$z = x++ - y;$ 结果 $z=2-3=-1, x=3$

$z = ++x - y;$ 结果 $z=3-3=0, x=3$

++x 的整体的值等于 x 加 1 以后的值

```
int x=2, y;
y = -(x++);
```

$y = -2 \quad x = 3$

例如:

①当++只与某个变量结合, 不与其它表达式混合运算时, 则++x 与 x++无区别, 都表示 $x=x+1$

x++;

```
int x=2, y;
y = (-x)++;
```

$y = -2$

$-x = -x + 1$

语法错误, 因为 -x 不是变量

[[Pasted image 20241201120334.png

(6) 逗号运算符和逗号表达式

逗号表达式的值等于最后一个表达式的值。

例如:

```
int x=0, y=2, z=3; 分隔符
z = f(x, y);
```

$x=8$

```
x = (y+=2, z++, y+=z); /* x=8 */
```

```
int x=0, y=2, z=3;
printf("%d %d", (x++, y+=x), z);
```

$y=4, x=4, z=8$

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
1 2 3 4 5 6
#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);
}
```

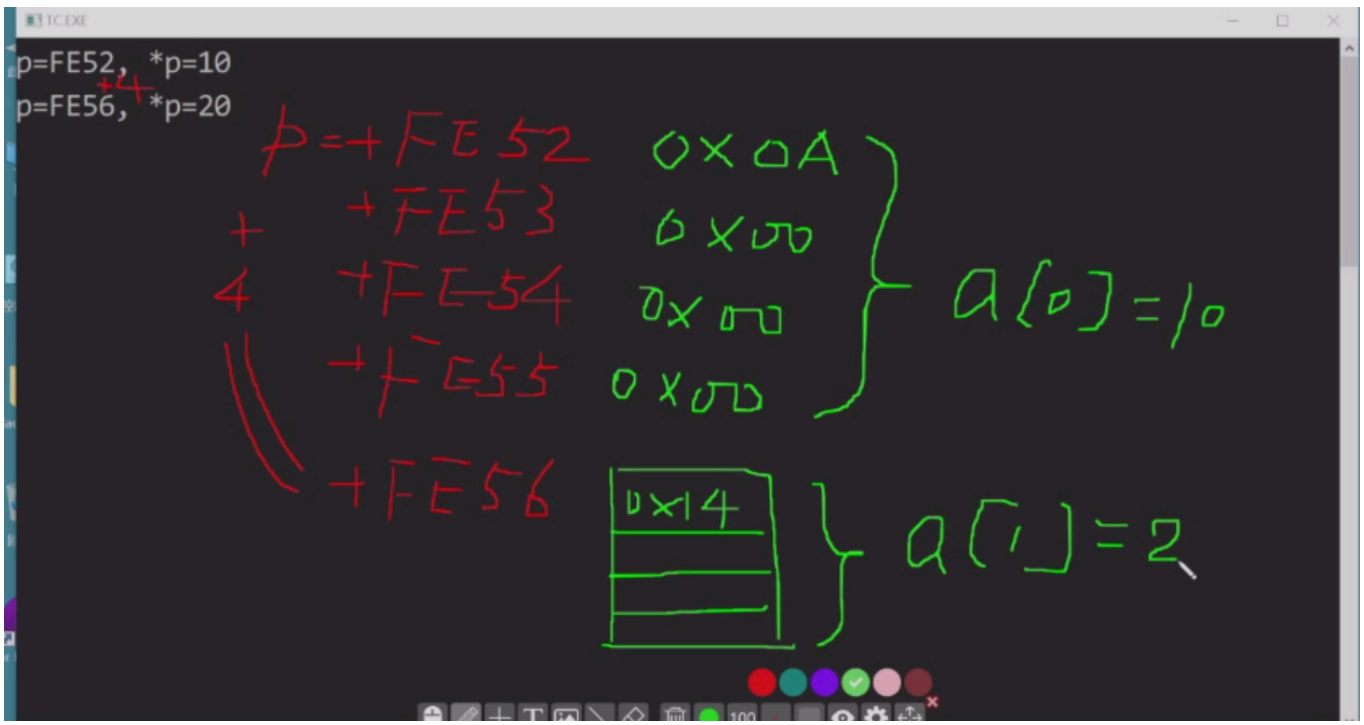
For Help, press F1

ln 9 col 36 11 20 PC ANSI 18:51 2024/12/4

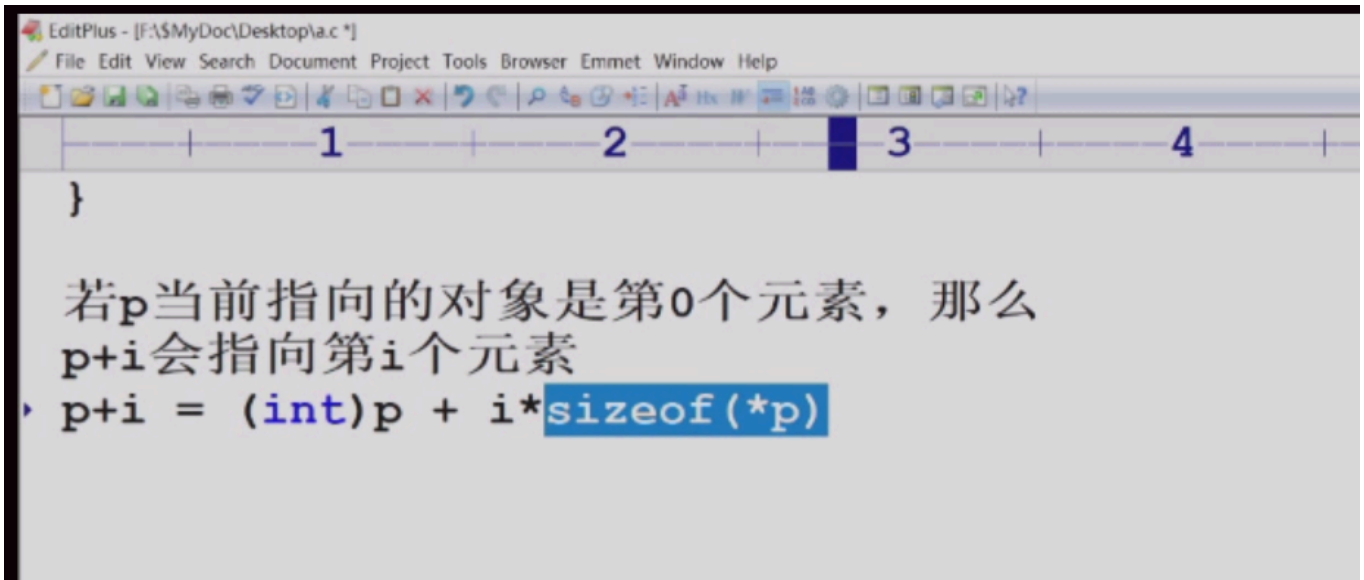
```
EditPlus - [F:\$MyDoc\Desktop\a.c]
File Edit View Search Document Project Tools Browser Emmet Window Help
1 2 3 4 5 6
#include <stdio.h>
main()
{
    long int a[100]={10, 20, 30};
    long int *p;
    p = &a[0];
    printf("p=%p, *p=%ld\n", p, *p);
    p++;
    printf("p=%p, *p=%ld\n", p, *p);
}
```

For Help, press F1

ln 9 col 36 11 20 PC ANSI 18:51 2024/12/4



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

$p+i = (\text{int})p + i * \text{sizeof}(*p)$

I

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

$q-p = ((\text{int})q - (\text{int})p) / \text{sizeof}(*p)$

指针作比较

指针输出时的扩充问题

```
pt.c x stdio.h x
1 #include<stdio.h>
2 #include<string.h>
3 int main()
4 {
5     long int a=0x12345678;
6     int i;
7     unsigned char *p;
8     p=(unsigned char *)&a;
9     for(int i=0;i<4;i++)
10    {
11        printf("p=%p, *p=%x\n", p+i, *(p+i));
12    }
13    return 0;
14 }
15
```

```
选定 D:\Users\27127\Desktop\程序\pta.exe
p=0061fec4, *p=78
p=0061fec5, *p=56
p=0061fec6, *p=34
p=0061fec7, *p=12
-----
Process exited after 0.2285 seconds with return value 0
请按任意键继续. . .
```

```
1 #include<stdio.h>
2 #include<string.h>
3 int main()
4 {
5     long int a=0x12345688;
6     int i;
7     char *p;
8     p=(char *)&a;
9     for(int i=0;i<4;i++)
10    {
11        printf("p=%p, *p=%x\n", p+i, *(p+i));
12    }
13    return 0;
14 }
15
```

```
D:\Users\27127\Desktop\程序\pta.exe
p=0061fec4, *p=ffffff88
p=0061fec5, *p=56
p=0061fec6, *p=34
p=0061fec7, *p=12
-----
Process exited after 0.2464 seconds with return value 0
请按任意键继续. . .
```

```
#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

永之 101

Internet Window Help

1 2 3 4 5 6

(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, 3); */
}

```

Handwritten notes:

- A box around `int *p;` in the function signature.
- Arrows pointing from `int *p;` to `int a[3]` in `main()`.
- A circle around `*(p+i)` with the equation $*(p+i) = a[i]$ written next to it.
- Another equation $p[0] = a[0]$ is written above the circle.

Taskbar: *p.c, For Help, press F1, 100, 18, 6216, FC, ANS

永之 101

Internet Window Help

1 2 3 4 5 6

```

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 */

```

Handwritten notes:

- Arrows pointing from `int a[3][4]` to the first two rows of the array.
- Annotations $p[1]+3$ and $2a[1][0]$ with arrows pointing to `p[1]` and `a[1][0]` respectively.
- Equation $a[1]$ written below `a[1]`.
- Equation $a[1]$ written below `a[1]`.
- Equation $a[1]$ written below `a[1]`.

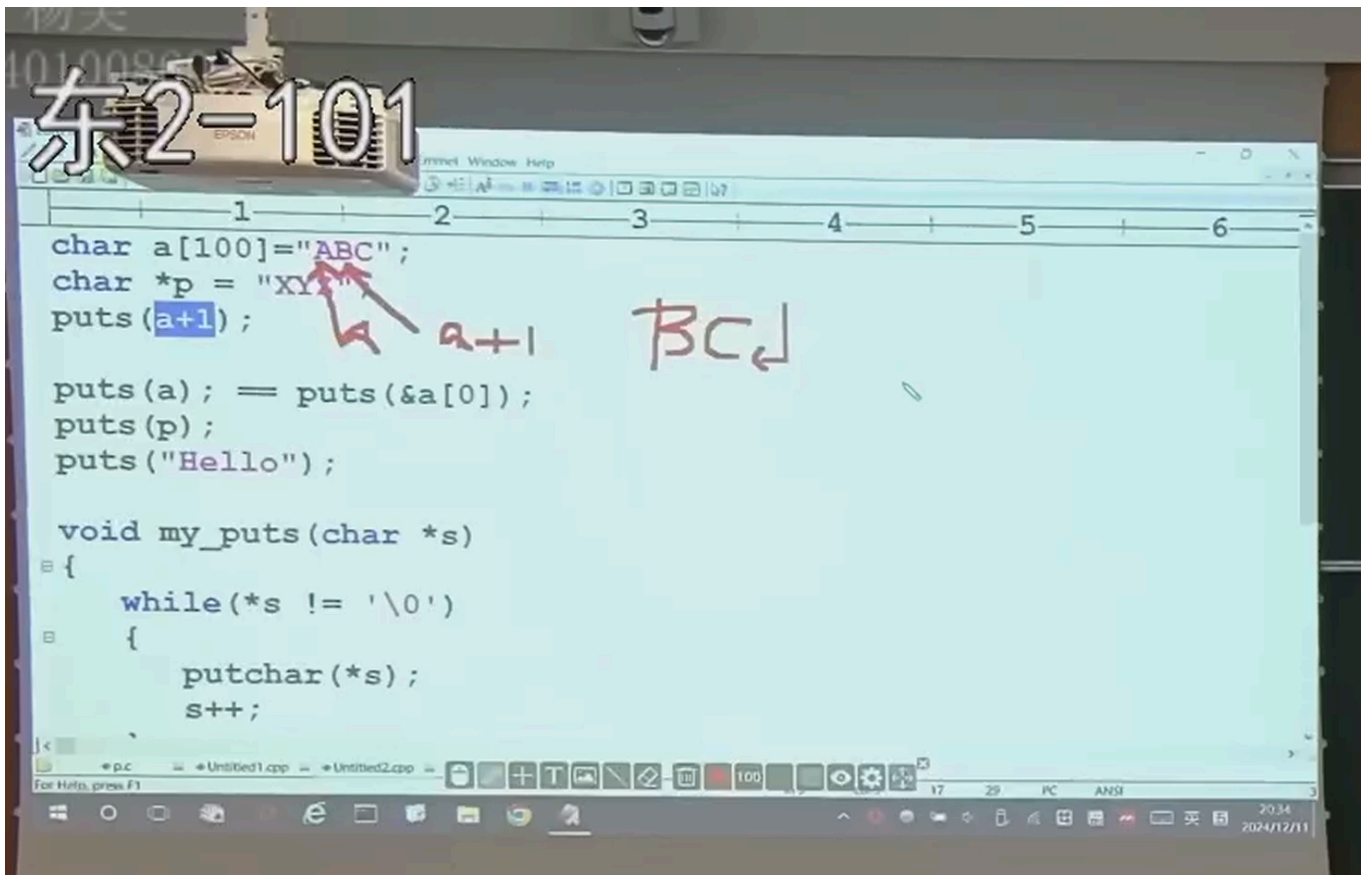
Taskbar: *p.c, For Help, press F1, 100, 11, 28, FC, ANS, 19:29, 2020/11/11

```
1 2 3 4
{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: I
*(*(p+i)+j)
*(p[i]+j)
(*(p+i))[j]
p[i][j]
|
y = *(q+1); /* y = a[0][1] = 1 */
```

```
1 2 3 4
char *p = "XYZ";
puts(a); == puts(&a[0]);
puts(p);
puts("Hello");

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

p.c Untitled1.cpp Untitled2.cpp 100



puts (a+1)会输出BC\n*

 **Note**

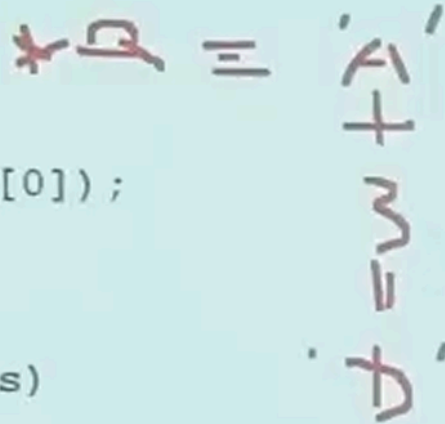
因为puts里面是地址

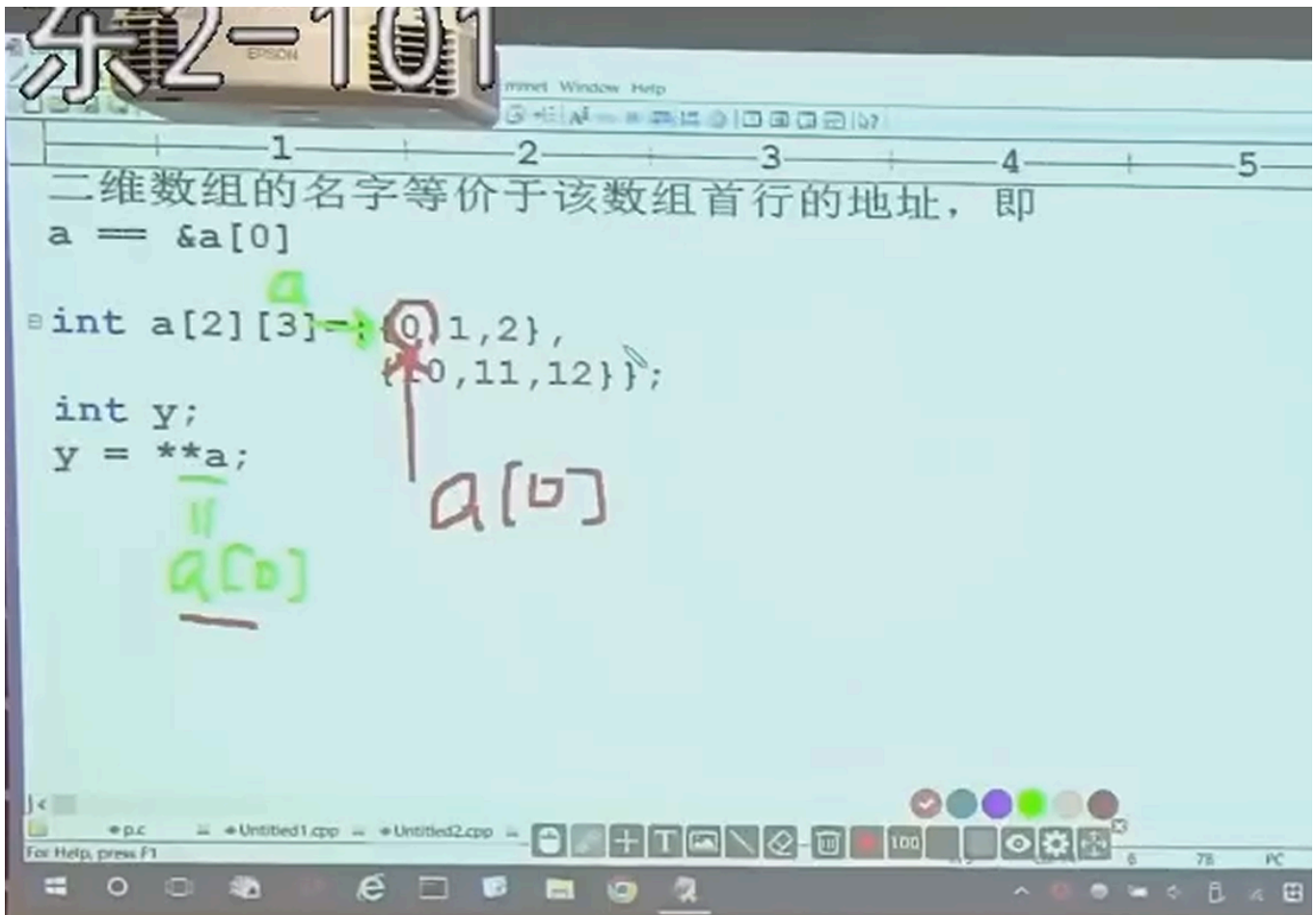
putchar里面不是地址, putchar (*a +3) 输出D

东2-101

```
char a[100]="ABC";  
char *p = "XY";  
puts(a+1);  
putchar(*a+3);  
  
puts(a); // puts(&a[0]);  
puts(p);  
puts("Hello");
```

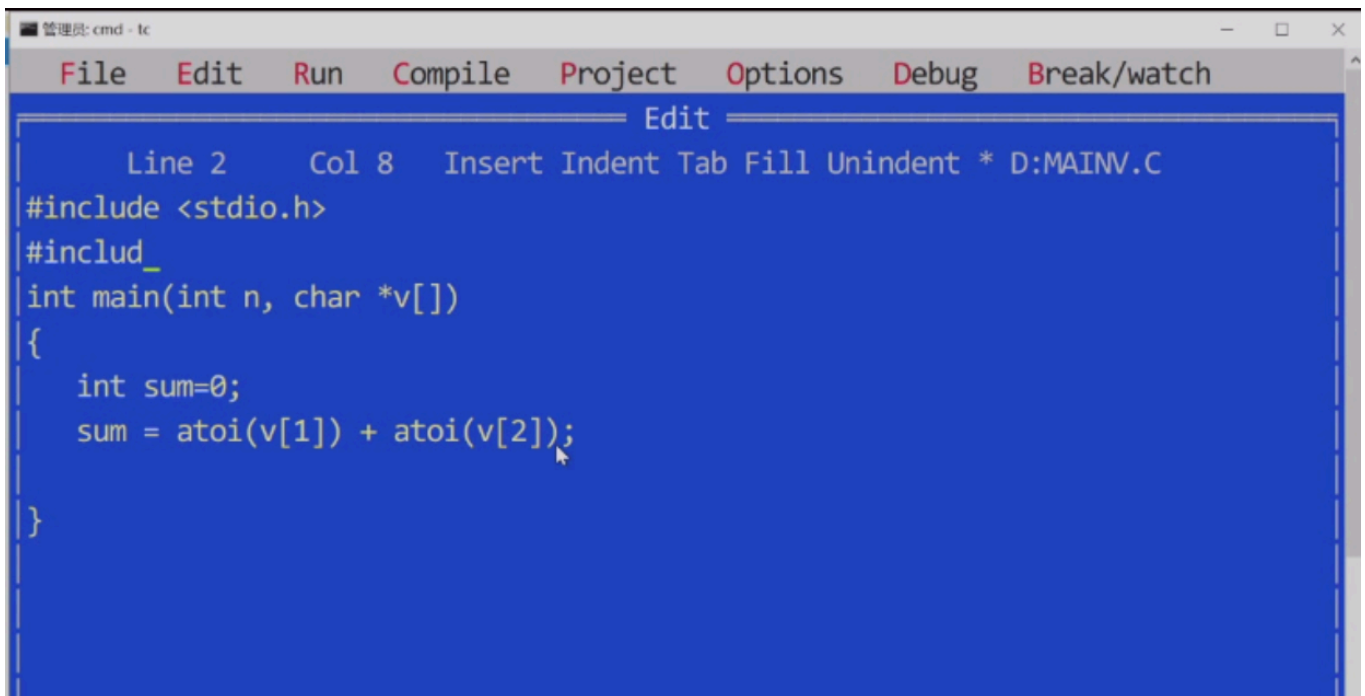
```
void my_puts(char *s)  
{  
    while(*s != '\0')  
    {  
        putchar(*s);  
    }  
}
```





`a`就是a【0】【0】**

命令行参数



结构体


```
1 | 2 | 3 | 4 | 5 | 6 |
char name[10];          char a[100];
int score;             a = "ABC"; 语法错误, 因为a==&a[0]是常数
};                     char a[100]="ABC"; 正确
main()                 char a[100];
{                       strcpy(a, "ABC"); 正确

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);
```

```
1 | 2 | 3 | 4 | 5 | 6 |
struct st
{
  char name[10];      char a[100];
  int score;          a = "ABC"; 语法错误, 因为a==&a[0]是常数
};                   char a[100]="ABC"; 正确
main()               char a[100];
{                     strcpy(a, "ABC"); 正确

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";
```

```
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);
}
```

Handwritten annotations:
- A box around {"Jerry", 60} with an arrow pointing to it from "p+i" above.
- "a(i)" written below the loop.

```
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);
}
```

Handwritten annotations:
- A box around `(*(p+i)).name` and another box around `(p+i)->name` with an arrow pointing from the first to the second.

```
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
```

杨昊

3240100860

东2-101

```
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
a等价于&a[0]
是常数, 非变量
(3) &a 与 &p 类型不同
char (*t)[100]; 此时t的类型就是&a的类型
char **
char a[100]="ABC";
char *p = a; 或 char *p = &a[0];
```



```
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 */
}

I
```

For help, press F1

22 00 PC ANSI 19:51 2024/12/18

东2-101

```
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;
```

```
ap p; 等价于char (*p) [10];

struct st
{
    char name[10];
    int score;
};
main()
{
    struct st a;    a.name
    scanf("%s %d", &a.name[0], &a.score);
}
```

注意!

&a是不对的, 但是程序跑出来是对的

变量

```
1 2 3 4 5 6
#include <stdio.h>
int n; /* n是全局变量，定义在任何函数之外 */ I
/* 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

```
#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++; | 100 |
}
main()
{
    n *= 10; h = | 1000 |
    f();
    printf("n=%d", n); h = | 100 |
}
```

```
#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; 内部优先
    n++;
}
main()
{
    n *= 10; n = | 1000 |
    f();
    printf("n=%d", n);
}
```



```
1 #include <stdio.h>
2 static int n=100;
3 void f(void){
4     int n=0;
5     n++;
6 }
7 int main(){
8     n*=10;
9     f();
10    printf("%d",n);
11    return 0;
12 }
```

D:\Users\27127\Desktop\hh.exe

1000

Process exited after 0.3019 seconds with return va
请按任意键继续. . .

东2-101

```
#include <stdio.h>

void f(void)
{
    n++;
}

int n= 100;
main()
{
    n*=10;
    f();
    printf("n=%d", n);
}
```

forward reference (向前引用)
上面的代码引用下面的变量或函数

5

会报错!

```
#include <stdio.h>
extern int n; /* 声明这个变量n在外面，即不在当前文件中 */

void f(void)
{
    n++;
}

int n = 100;
main()
{
    n *= 10;
    f();
    printf("n=%d", n);
}
```

forward reference (向前引用)
上面的代码引用下面的变量或函数

欺骗编译器n在外面（其实在里面），这样就不报错了

```
/* 保存为f.c */
int n = 100; /* 定义了一个全局变量n */
/* 全局静态变量只能在本文件内引用，不能跨文件 */
/* 全局变量可以跨文件引用 */

void f(void)
{
    n++;
}
```

```
1  #include <stdio.h>
2
3  int main(){
4      n*=10;
5      printf("%d",n);
6      static int n=100;
7      return 0;
8  }//不可以//
9  int main(){
10     static int n;
11     n*=10;
12     printf("%d",n);
13     n=100;
14     return 0;
15 }//可以//
16 int main(){
17     int n ;
18     n*=10;
19     printf("%d",n);
20     n=100;
21     return 0;
22 }//不可以//
23
24
25
26
27
```



```
MinGW GCC10.2.0 32-bit Debug
hh.c x main.cpp [*] x
1 #include <stdio.h>
2
3 int main(){
4     static int n;
5     n*=10;
6     n=100;
7     printf("%d",n);
8
9 }//可以//
10
11
12
```

```
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

```
1 2 3 4 5 6
#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)
    {
        . . . . .
        . . . . .
    }
}
```

For Help, press F1

26 SEFB PC ANS 20:43 2024/12

```
1 2 3 4 5 6
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 */
}
```

6

```
1      puts("Cannot open file!");
2      exit(0); /* 强制结束程序的运行 */
3  }
4  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 x
1 #include <stdio.h>
2
3 int main(){
4     FILE *fp,*fp2;
5     fp=fopen("abc.txt","r");
6     fp2=fopen("xyz.txt","w");
7     if(fp==NULL||fp2==NULL){
8         puts("wrong!");
9         exit(0);
10    }
11    while(!feof(fp)){
12        char c=fgetc(fp);
13        fputc(c,fp2);
14    }
15    fclose(fp);
16    fclose(fp2);
17
18 }
19
```

文件 编辑

ABC
[]

```
1 #include <stdio.h>
2
3 int main(){
4     FILE *fp,*fp2;
5     fp=fopen("abc.txt","r");
6     fp2=fopen("xyz.txt","w");
7     if(fp==NULL||fp2==NULL){
8         puts("wrong!");
9         exit(0);
10    }
11    char c;
12    while((c=fgetc(fp))!=EOF){
13
14        fputc(c,fp2);
15    }
16    fclose(fp);
17    fclose(fp2);
18
19 }
20
```

文件 编辑

ABC