C程序设计基础

Introduction to C programming

Lecture 12: I/O

张振国 zhangzg@sustech.edu.cn

南方科技大学/理学院/地球与空间科学系

Review on L11 Pointer II

Pointer and Array

Pointer and function

Memory management(advanced uses)

Pointer points to array

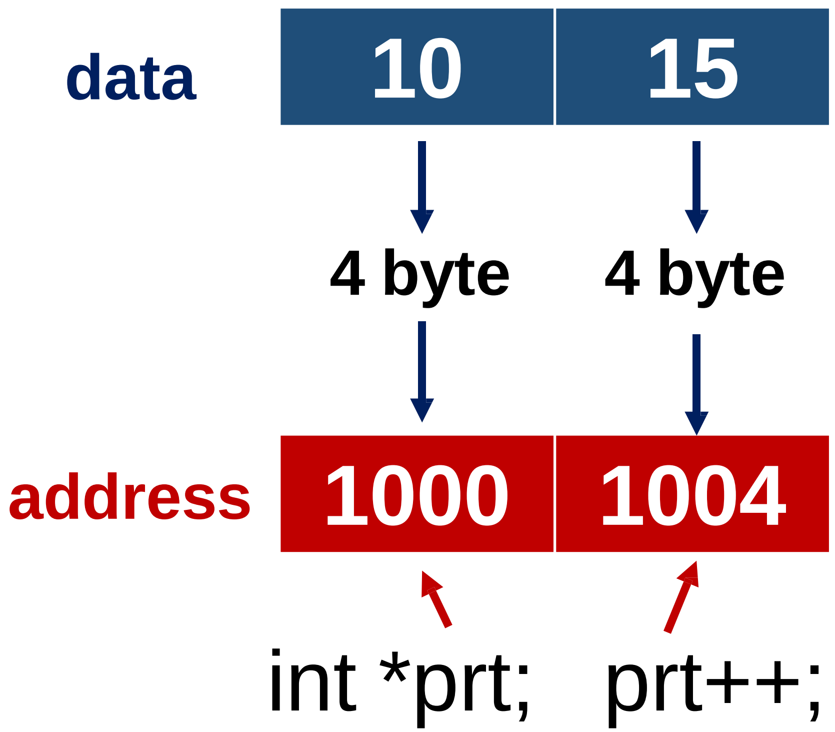
```
int a[3]={1,2,3};

int *b = a;
or
int *b = &a[0];
```

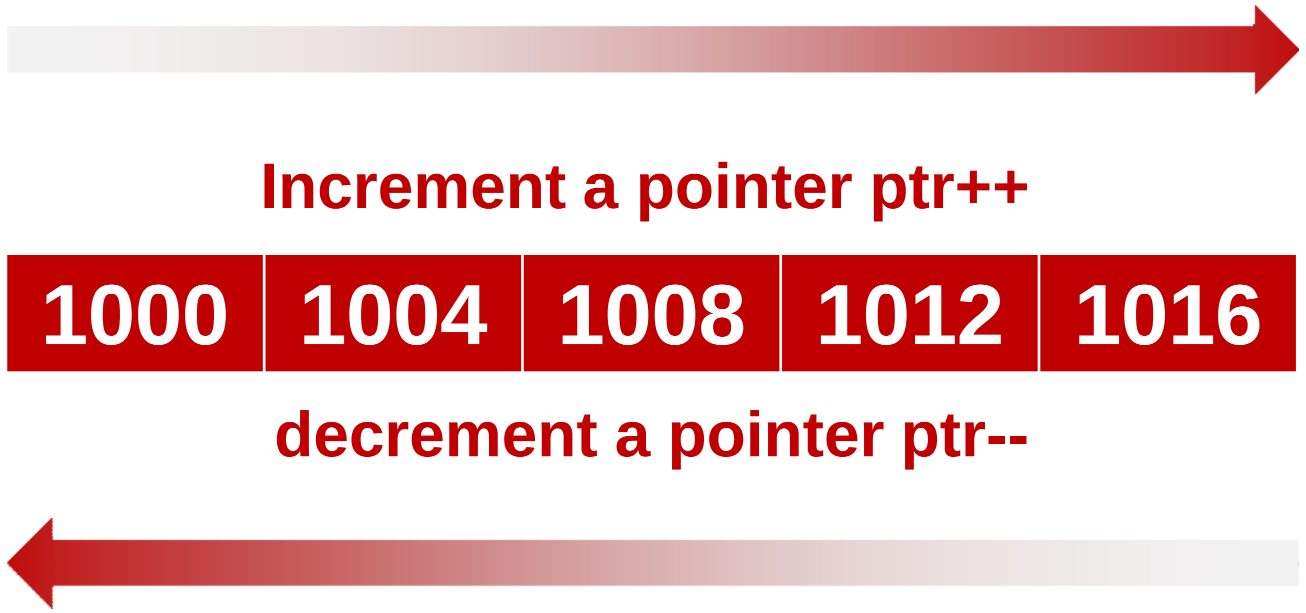
Array	Address	Content
a[0]	17d8f780	1
a[1]	17d8f784	2
a[2]	17d8f788	3
…	…	

Address of the first element is assigned to pointer

Pointer points to array

Four arithmetic operators that can be used on pointers: ++, --, +, -

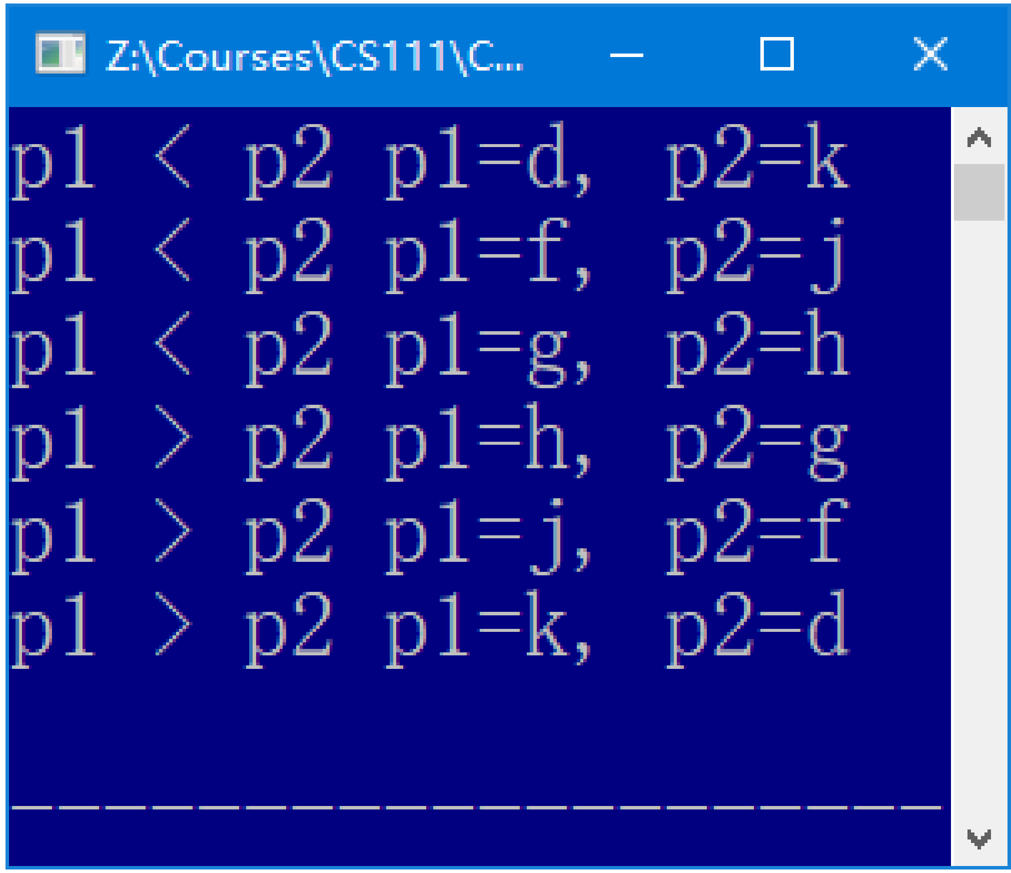




Pointer points to array

Use pointer to compare memory address: >, <, ==

```c
#include <stdio.h>
main()
{
    char str[7] = "dfghjk", * p1, * p2;
    p1 = str;
    p2 = p1 + 5;

    for (int i = 0; i < 6; i++)
    {
        if (p1 < p2)
        {
            printf("p1 < p2\t");
            printf("p1=%c, p2=%c\n", *p1, *p2);
        }
        else {
            printf("p1 > p2\t");
            printf("p1=%c, p2=%c\n", *p1, *p2);
        }
         *p1++;    //p1++
         *p2--;    //p2--
    }
}
```



```
p1 < p2 p1=d, p2=k
p1 < p2 p1=f, p2=j
p1 < p2 p1=g, p2=h
p1 > p2 p1=h, p2=g
p1 > p2 p1=j, p2=f
p1 > p2 p1=k, p2=d
```

L10_compare.c

Pointer points to array

* and ++/--

数组用法

```
a[i++]=j;
```

指针用法??

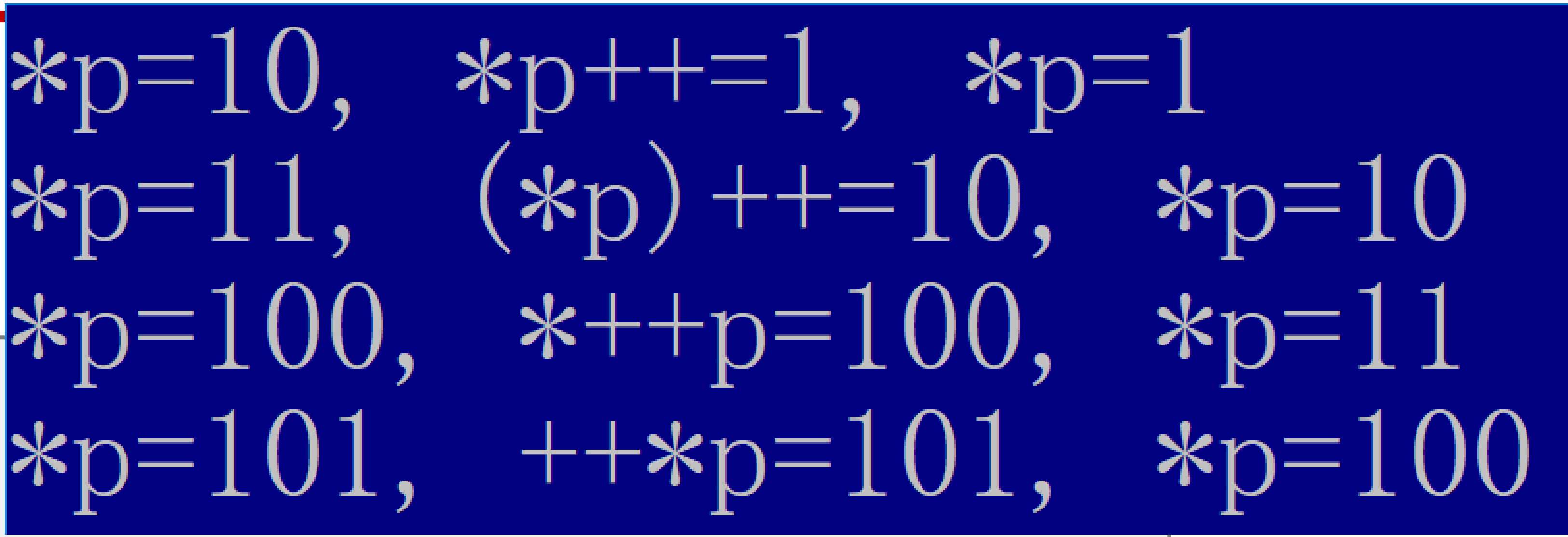
```
p = a;

*p++=j;        ✓

(*p)++=j;      ✗

*(p++)=j;      ✓
```

Pointer points to array

* and ++/--

`*p++ or *p(++)`	先使用p(表达式取指针值)，后移动指针(自增)
`(*p)++`	先使用p(表达式取指针值)，后自增指针指向的值
`*++p or *(++p)`	先移动指针使用(自增)，后使用(取值)
`++*p or ++(*p)`	先取值，后自增指针指向的值

Pointer points to array



```
*p=10,  *p++=1,  *p=1
*p=11,  (*p)++=10,  *p=10
*p=100,  *++p=100,  *p=11
*p=101,  ++*p=101,  *p=100
```

```
#include <stdio.h>

int main(void) {
     int a[4] = {1, 10, 100, 1000};
     int *p = a;

     printf("*p=%d, *p++=%d, *p=%d\n",   *p, *p++,   *p);
     printf("*p=%d, (*p)++=%d, *p=%d\n", *p, (*p)++, *p);
     printf("*p=%d, *++p=%d, *p=%d\n",   *p,  *++p,  *p);
     printf("*p=%d, ++*p=%d, *p=%d\n",   *p,  ++*p,  *p);
}
```

Pointer points to array

输出数组元素的三种方法：

下标法

```
void  main() {
        int a[10];
        int i;
        for (i = 0; i < 10; i++)
                scanf("%d", &a[i]);
        printf("\n");
        for (i = 0; i < 10; i++)
                printf("%d", a[i]);
}
```

通过数组名计算数组元素地址，找出元素的值

```
void  main() {
        int a[10];
        int i;
        for (i = 0; i < 10; i++)
                scanf("%d", &a[i]);
        printf("\n");
        for (i = 0; i < 10; i++)
                printf("%d",*(a+i));
}
```

用指针变量指向数组元素

```
void  main() {
        int a[10];
        int *p, i;
        for (i = 0; i < 10; i++)
                scanf("%d", &a[i]);
        printf("\n");
        for (p = a; p < (a + 10); p++)
                printf("%d", *p);
}
```

Pointer points to 2D/ND array

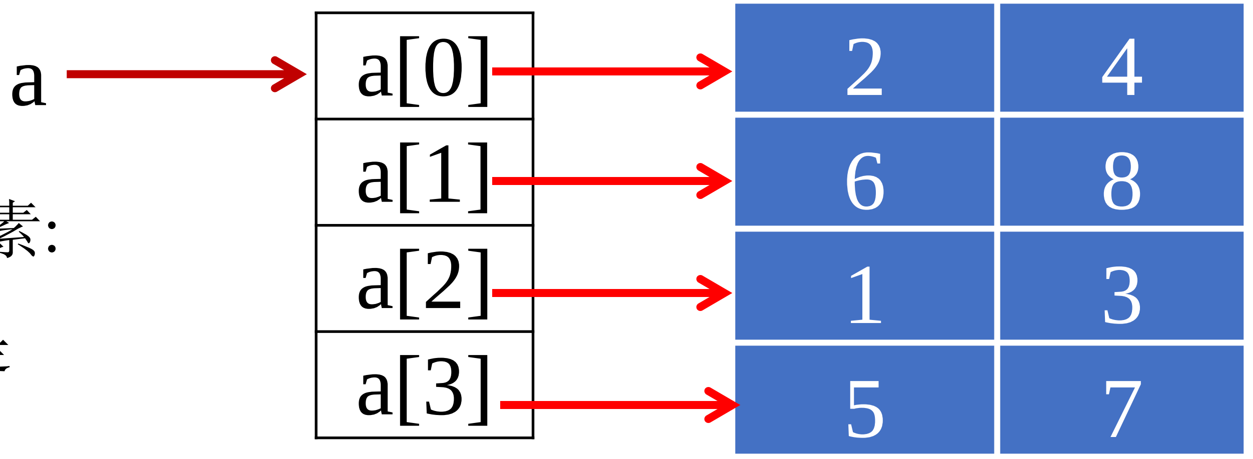
```c
#include <stdio.h>

int main(void) {
     int *p;
     int a[4][2]={{2,4},{6,8},{1,3},{5,7}};

     p = a;
     printf("Value of p: %d\n", *p);
}
```

2	4
6	8
1	3
5	7

[错误] 无法转换 'int [3][2]' 到 'int*' 在赋值时

Pointer points to 2D array

```
int a[4][2]={{2,4},{6,8},{1,3},{5,7}};
```



a是二维数组名。a数组包含4行，即4个行元素: a[0]，a[1]，a[2]，a[3]。而每一个行元素又是一个一维数组，它包含2个元素（即2个列元素）。可以认为二维数组是“数组的数组”，即二维数组a有4个一维数组构成

a[0]、a[1]、a[2]、a[3]分别表示为一维数组

Pointer points to 2D array

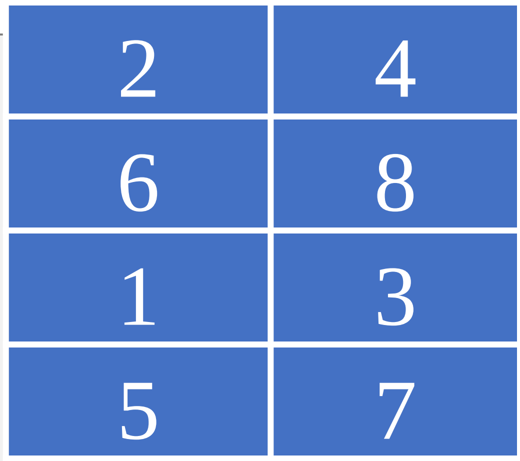
```
int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,21,23}}
```

表示形式	含义	值
a	二维数组名，指向一维数组a[0]，即0行起始地址	2000
a[0],*(a+0),*a	0行0列元素地址	2000
a+1,&a[1]	1行起始地址	2016
a[1],*(a+1)	1行0列元素a[1][0]的地址	2016
a[1]+2,*(a+1)+2,&a[1][2]	1行2列元素a[1][2]的地址	2024
(a[1]+2),(*(a+1)+2),a[1][2]	1行2列元素a[1][2]的值	元素值，13

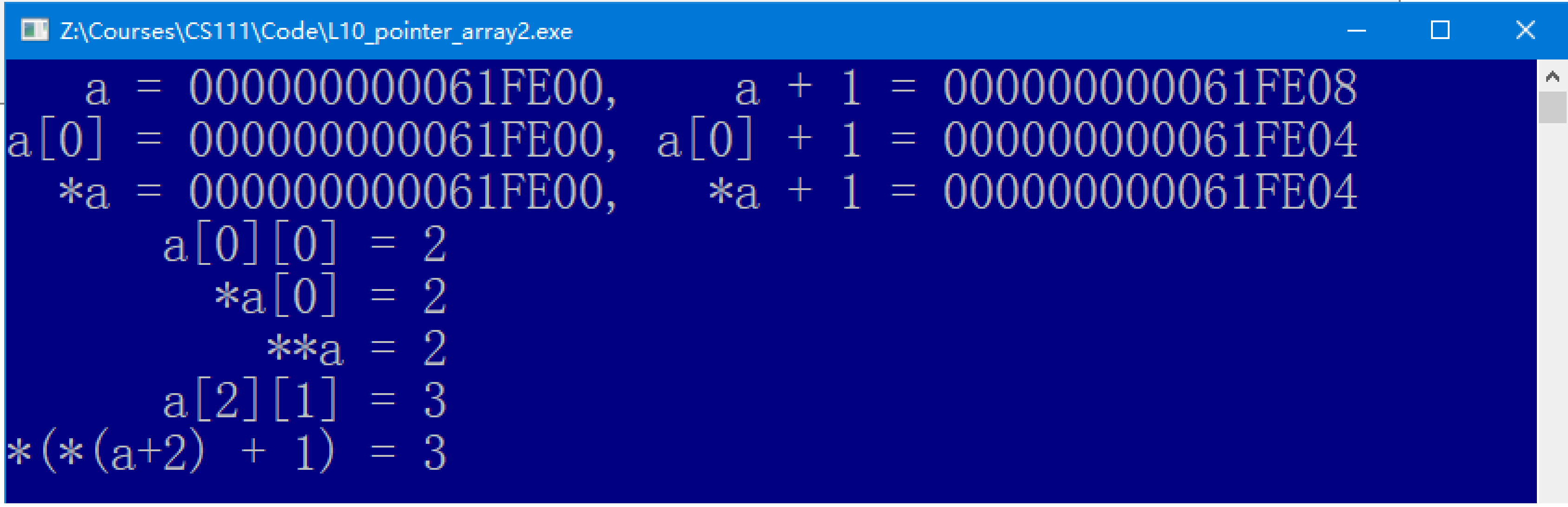
注意：
- ☐ **二维数组名(如a)是指向行(一维数组)的**。**一维数组名（如a[0]）是指向列元素的。**在指向行的指针前面加一个*，就转换为指向列的指针。反之，在指向列的指针前面加&，就成为指向行的指针。
- ☐ 不要把&a[i]简单地理解为a[i]元素的存储单元的地址，它只是一种**地址**的计算方法，能得到第i行的起始地址。
- ☐ &a[i]和a[i]的值是一样的，但它们的基类型是不同的。&a[i]或a+i指向行，而 a[i]或*(a+i)指向列。

Pointer points to 2D array

```c
int main(void) {
	int a[4][2] = { { 2, 4 }, { 6, 8 }, { 1, 3 },{ 5, 7 }};
	printf("   a = %p,     a + 1 = %p\n", a, a + 1);
	printf("a[0] = %p, a[0] + 1 = %p\n", a[0], a[0] + 1);
	printf("  *a = %p,    *a + 1 = %p\n", *a, *a + 1);
	printf("      a[0][0] = %d\n", a[0][0]);
	printf("        *a[0] = %d\n", *a[0]);
	printf("          **a = %d\n", **a);
	printf("      a[2][1] = %d\n", a[2][1]);
	printf("*(*(a+2) + 1) = %d\n", *(*(a + 2) + 1));
	return 0;
}
```



2	4
6	8
1	3
5	7



```
   a = 000000000061FE00,     a + 1 = 000000000061FE08
a[0] = 000000000061FE00, a[0] + 1 = 000000000061FE04
  *a = 000000000061FE00,    *a + 1 = 000000000061FE04
      a[0][0] = 2
        *a[0] = 2
          **a = 2
      a[2][1] = 3
*(*(a+2) + 1) = 3
```

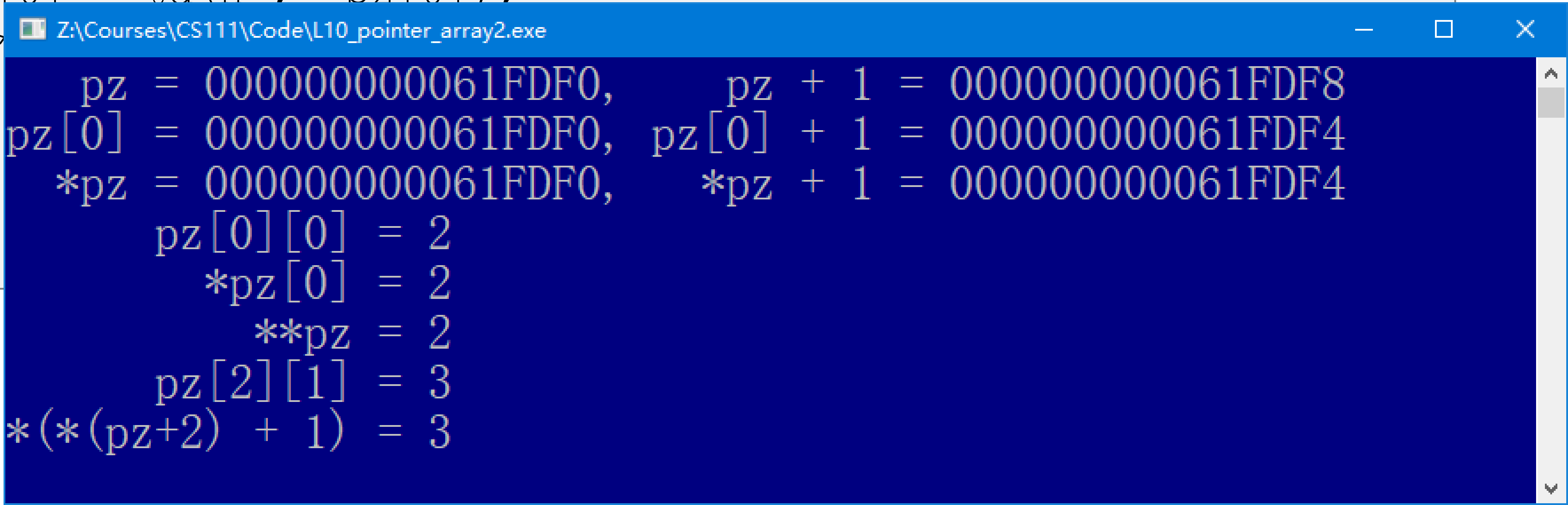
Pointer points to 2D array

声明指向数组的指针：（数组指针）

type (*name)[2]

2	4
6	8
1	3
5	7

```
int main(void) {
        int a[4][2] = { { 2, 4 }, { 6, 8 }, { 1, 3 }, { 5, 7 } };
        int(*pz)[2];
        pz = a;
        printf("    pz = %p,     pz + 1 = %p\n", pz, pz + 1);
        printf("pz[0] = %p, pz[0] + 1 = %p\n", pz[0], pz[0] + 1);
        printf("   *pz = %p,    *pz + 1 = %p\n", *pz, *pz + 1);
        printf("       pz[0][0] = %d\n", pz[0][0]);
        printf("         *pz[0] = %d\n", *pz[0]);
        printf("            **
        printf("       pz[2]
        printf("*(*(pz+2) +
        return 0;
}
```


Z:\Courses\CS111\Code\L10_pointer_array2.exe

```
    pz = 000000000061FDF0,     pz + 1 = 000000000061FDF8
pz[0] = 000000000061FDF0, pz[0] + 1 = 000000000061FDF4
   *pz = 000000000061FDF0,    *pz + 1 = 000000000061FDF4
       pz[0][0] = 2
         *pz[0] = 2
           **pz = 2
       pz[2][1] = 3
*(*(pz+2) + 1) = 3
```

Pointer points to 2D array

声明指向数组的指针：（**数组指针**）

`type (*name)[2]`

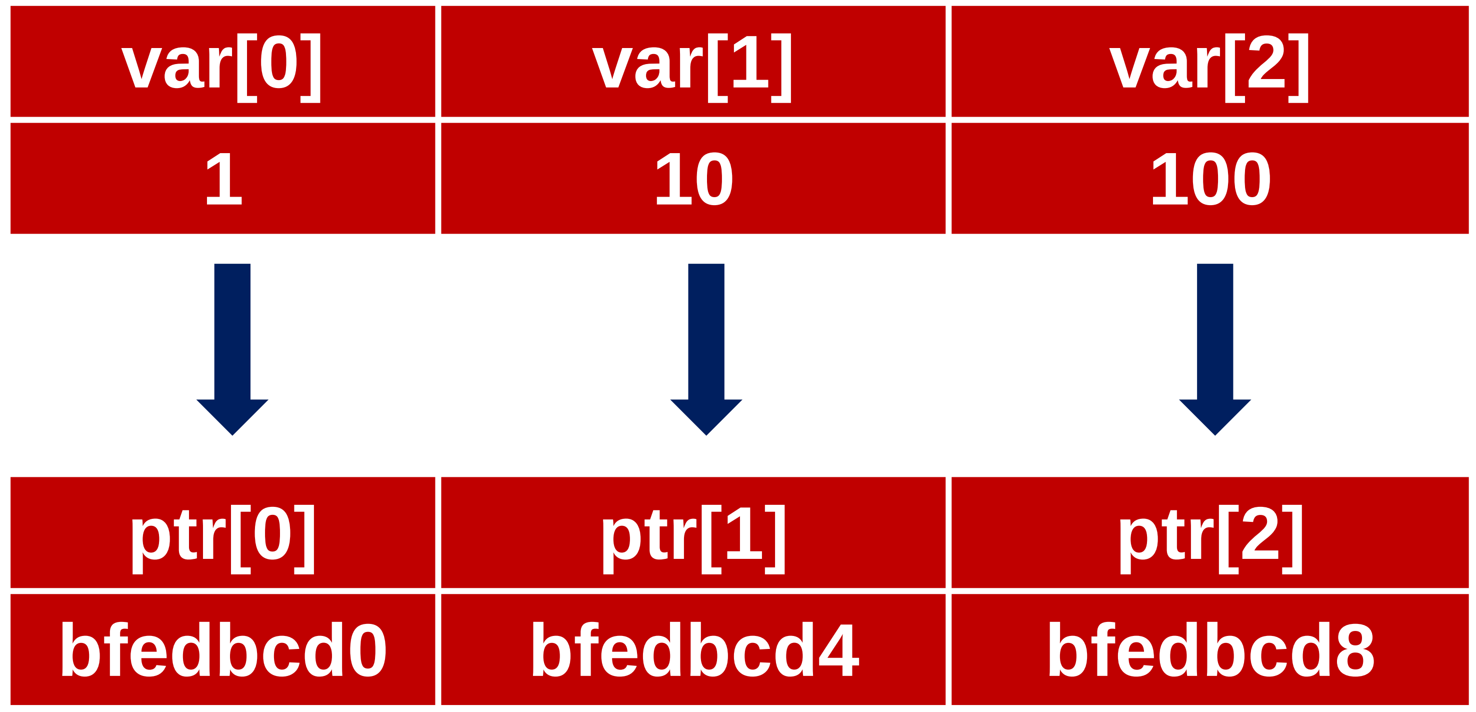
```
int main(void) {
        int a[4][2] = { { 2, 4 }, { 6, 8 }, { 1, 3 }, { 5, 7 } };
        int(*pz)[2];
}
```

2	4
6	8
1	3
5	7

- ✓ 此处[2]表示列的元素是2个，pz++一次移动2个元素；**可以为其他数据，语法上没有错误**
- ✓ int(*pz)[4]可以将原始数组重定义为2*4的矩阵，pz++一次移动4个元素

Array of pointers指针数组

An array to store pointers



int val[3];

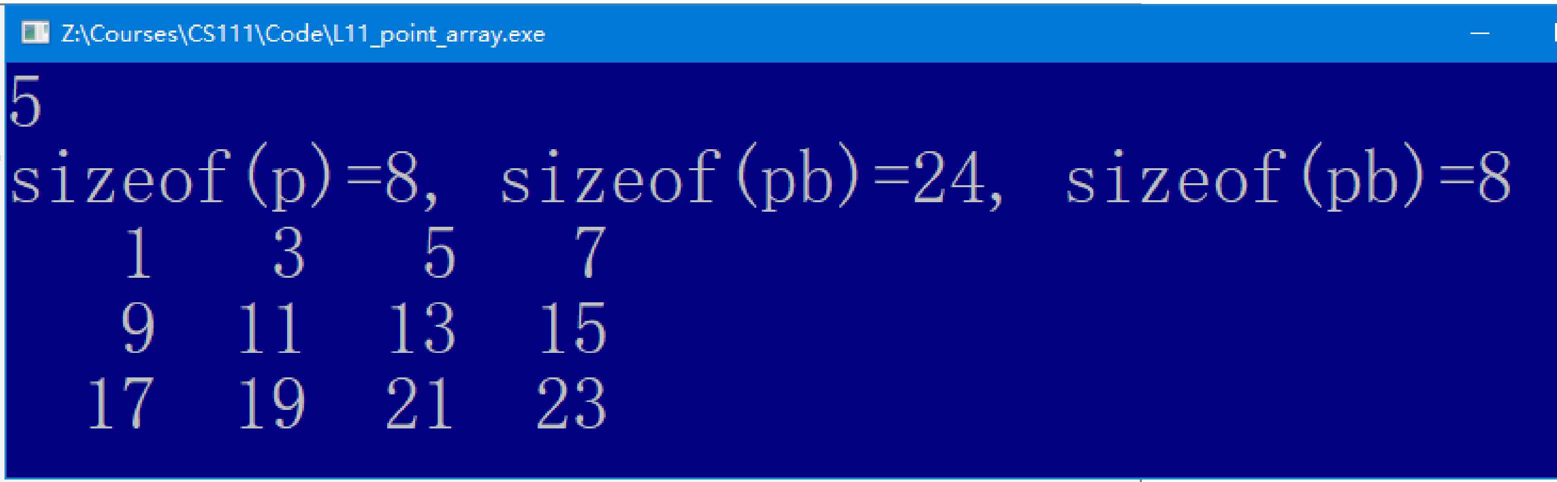
var[0]	var[1]	var[2]
1	10	100

int *ptr[3];

ptr[0]	ptr[1]	ptr[2]
bfedbcd0	bfedbcd4	bfedbcd8

Pointer points to 2D array

```c
#include <stdio.h>
int main(void) {
      int a[3][4] = {1, 3, 5, 7
      int *p, *pa[3], (*pb)[4];
      pb = a;
      pa[0] = a[0];
      pa[1] = a[1];
      pa[2] = a[2];
      printf("%d \n", *(pa[0] + 2 ));
      printf("sizeof(p)=%d, sizeof(pb)=%d, sizeof(pb)=%d", \
             sizeof(p), sizeof(pa), sizeof(pb));
      for (p = a[0]; p < a[0] + 12; p++) {
            if ((p - a[0]) % 4 == 0)
                  printf("\n");
            printf("%4d", *p);
      }
      printf("\n");
      return 0;}
```



```
5
sizeof(p)=8, sizeof(pb)=24, sizeof(pb)=8
   1   3   5   7
   9  11  13  15
  17  19  21  23
```

Content

function、array and pointer

声明数组形参：

数组名是该数组首元素的地址，作为实际参数的数组名要求形式参数是一个与之匹配的指针。

下面两种形式的函数定义等价：

```
int sum(int *ar,int n)
{ ...
}

int sum(int ar[],int n)
{ ...
}
```

function、array and pointer

假设flizny是一个数组：

flizny == &flizny[0]

数组名即为元素的首地址。

```
int sum(int *ar) {
    int i;
    int total = 0;
    for (i = 0; i < 10; i++) {
        total += ar[i];
    }
    return total;
}
```

int *ar 和int ar[] 的形式都表示ar是一个指向int 的指针!!!

function、array and pointer

如果有一个实参数组，要想在函数中改变此数组中的元素的值，实参与形参的对应关系：

形参和实参都用数组名

```
int main() {
        int a[10];
        ...
        f(a, 10);
        ...
}

int f(int x[], int n) {
        ...
}
```

实参用数组名，形参用指针变量

```
int main() {
        int a[10];
        ...
        f(a, 10);
        ...
}

int f(int *x, int n) {
        ...
}
```

形参和实参都用指针变量

```
int main() {
        int a[10], *p = a;
        ...
        f(p, 10);
        ...
}

int f(int *x, int n) {
        ...
}
```

实参为指针变量，形参为数组名

```
int main() {
        int a[10], *p = a;
        ...
        f(p, 10);
        ...
}

int f(int x[], int n) {
        ...
}
```

Function can return pointer

一个函数可以带回一个整型值、字符值、实型值等，也可以带回指针型的数据，即地址。其概念与以前类似，只是带回的值的类型是指针类型而已。

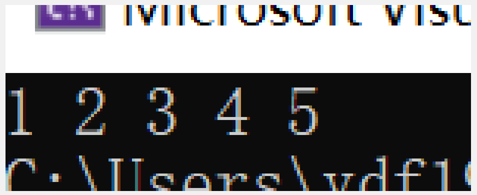
一般定义形式为：

类型名 *函数名（参数列表）

Function can return pointer

```
int * myFunction()
{

 . . .

}
```

```
int* merge(int a, int b, int c, int d, int e)
{
int* array = (int*)malloc(sizeof(int) * 5);
array[0] = a;
array[1] = b;
array[2] = c;
array[3] = d;
array[4] = e;
return array;
}

int main()
{
int* array = merge(1, 2, 3, 4, 5);
for (int i = 0; i < 5; i++)
printf("%d ", array[i]);
return 0;
}
```



```
1 2 3 4 5
```

Pointer of a function

指向函数的指针

- 用指针变量可以指向一个函数。
- 函数在编译时被分配给一个入口地址，函数名代表函数的起始地址。这个函数的入口地址就称为**函数的指针**。

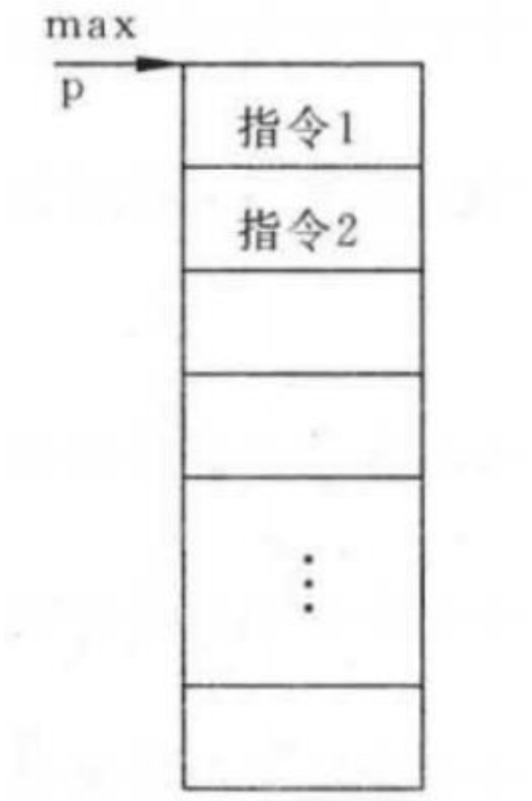
```
int (*p)(int, int);
```

指向函数类型为整形，且有两个整型参数的函数

Pointer of a function

利用指针变量调用它所指向的函数

```
int main(void) {
        int max(int, int);
        int(*p)(int, int);
        int a, b, c;
        p = max;    //只给出函数名不需要参数
        printf("please enter a and b:");
        scanf("%d %d", &a, &b);
        c = (*p)(a, b);
        printf("a=%d\nb=%d\nmax=%d\n", a, b, c);
        return 0;
}
int max(int x, int y) {
        int z;
        if (x > y)  z = x;
        else   z = y;
        return z;
}
```

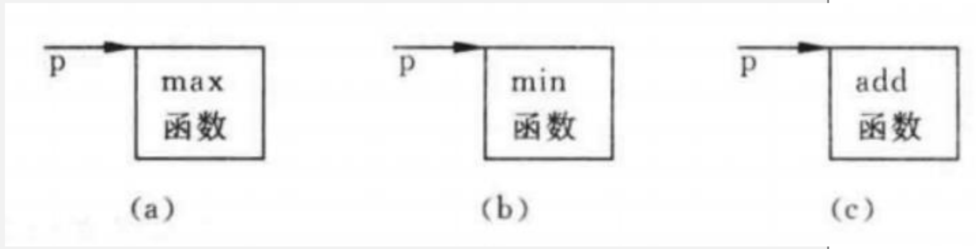


指向函数的指针变量**不能**
进行算数运算（p+n）！！

Pointer of a function

```
#include <stdio.h>

void main() {
    int max(int, int);
    int min(int, int);
    int add(int, int);
    void process (int, int, int(*fun)());
    int a, b;
    printf("enter a and b:");
    scanf("%d %d", &a, &b);
    printf("max=");
    process(a, b, max);
    printf("min=");
    process(a, b, min);
    printf("sum=");
    process(a, b, add);
}
```

```
int max(int x, int y) {
    return x > y ? x : y;
}
int min(int x, int y) {
    return x < y ? x : y;
}
int add(int x, int y) {
    int z;
    z = x + y;
    return z;
}

void process(int x, int y, int(*fun)(int, int))
{
    int result;
    result = (*fun)(x, y);
    printf("%d\n", result);
}
```



Character pointer

用字符数组存放一个字符串，
然后输出该字符串

```
int main(void) {
        char string[] = "I love China!";
        printf("%s\n", string);
        return 0;
}
```

用字符指针存放一个字符串，
然后输出该字符串

```
int main(void) {
        char *string = "I love China!";
        printf("%s\n", string);
        return 0;
}
```

Character pointer

```
int main(void) {
        char *string = "I love China!";
        printf("%s\n", string);
        return 0;
}
```

对字符指针变量string初始化,实际上是**把字符串第1个元素的地址(即存放字符串的字符数组的首元素地址)赋给指针变量string**,使 string指向字符串的第1个字符,由于字符串常量"I love China!"已由系统分配在内存中连续的14个字节中,因此, string 就指向了该字符串的第一个字符。

```
char *string;
*srting = "I love China!"
```

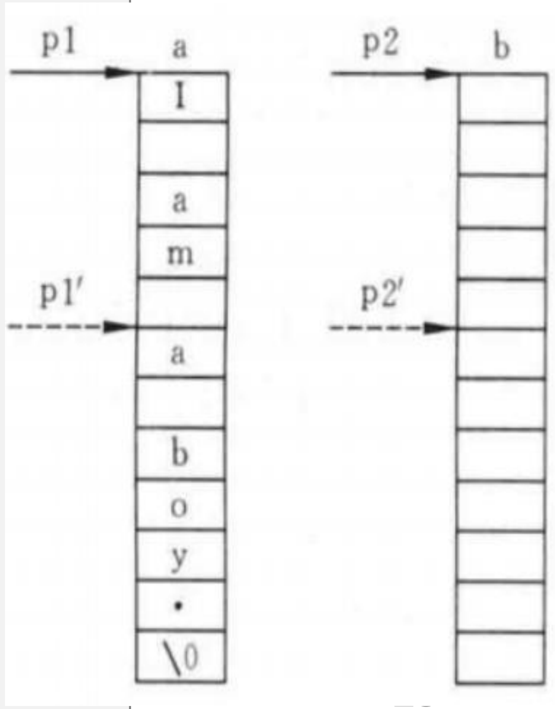
```
char *string;
srting = "I love China!"
```

Character pointer

将字符串a复制为字符串b,然后输出字符串b。

```
#include<stdio.h>
int main(void) {
        char a[] = "I am a student.", b[20];
        int i;
        for (i = 0; * (a + i) != '\0'; i++)
                *(b + i) = *(a + i);
        *(b + i) = '\0';
        printf("string a is: %s\n", a);
        printf("srting b is:");
        for (i = 0; b[i] != '\0'; i++)
                printf("%c", b[i]);
        printf("\n");
        return 0;
}
```

```
#include<stdio.h>
int main(void) {
        char a[] = "I am a student.", b[20];
        char *p1, *p2;
        p1 = a;
        p2 = b;
        for (; *p1 != '\0'; p1++, p2++)
                *p2 = *p1;
        *p2 = '\0';
        printf("string a is: %s\n", a);
        printf("string b is: %s", b );
        return 0;
}
```



Character pointer

字符指针变量 VS 字符数组

1.字符数组由若干个元素组成，每个元素中放一个字符，而字符指针变量存放的是地址（字符串第1个字符的地址），决不是将字符串放到字符指针变量中。

2.赋值方式。**可以对字符指针变量赋值，但不能对数组名赋值。**

```
char *a;
a = "I love China!";//合法赋给a的不是字符串而是第一个元素的地址

char  str[14];
str＝"I love China"; //非法！数组名是地址不是常量，不能被赋值
```

3.初始化含义。

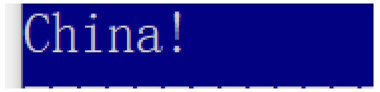
```
char *a;
a = "I love China!";//把字符串第一个元素的地址赋给a

char  str[14]="I love China!";//定义字符数组str，并把字符串赋值给数组中各个元素
```

Character pointer

4.存储单元的内容。编译时为字符数组分配若干存储单元，以存放各元素的值，而对字符指针变量，只分配了一个存储单元。

5.指针变量的值是可以改变的，而字符数组名代表一个固定的值（数组首元素的地址），不能改变。

```
char *a = "I love China!";
a = a + 7;
printf("%s", a);
```



```
China!
```

6.字符数组中各元素的值是可以改变的（可以对它们再赋值），但字符指针变量指向的字符串常量中的内容是不可以被取代的（不能对它们再赋值）。

```
char *a = "House";
a[2] = ‘r’;              //非法，字符串常量不能改变
printf(“%c\n”, a[2]);    //但可以单独输出
```

void pointer

- C99允许使用基类型为 void 的指针类型。
- 可以定义一个基类型为void 的指针变量(即void *型变量)，它不指向任何类型的数据。可以理解为“指向空类型”或“不指向确定的类型”的数据，只提供一个纯地址。
- 它可以用来指向一个抽象的类型的数据，在将它的值赋给另一指针变量时要进行强制类型转换使之适合于被赋值的变量的类型。

```
char *p1;
void *p2;
p1 = (char *)p2;
```

- 主要应用于调用动态存储分配函数时出现。如：

```
pt=(int *)malloc(100);
```

- 也可以应用于函数中：

```
void *fun(char ch1, char ch2)
p1 = (char*)fun(ch1,ch2);
```

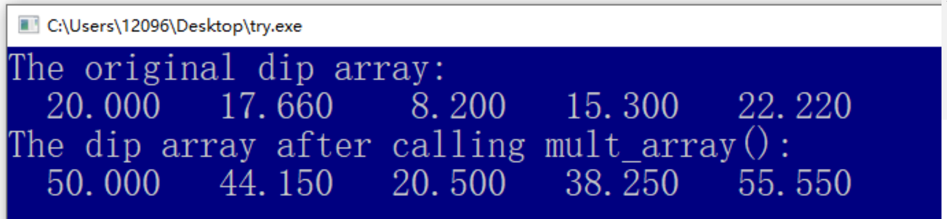
const

编写处理基本类型（如int）的函数时，要选择传递int类型的值还是传递指向int的指针。对于数组来说，必须传递指针，这样效率高。但是传递指针会导致函数可能会将原始数据修改，因此可以在函数原型和函数定义中声明形式参数时使用关键字const。如：

```
int sum(const int ar[],int n);
```

```
#include <stdio.h>
#define SIZE 5
void show_array(const double ar[], int n);//不改变数组
void mult_array(double ar[], int n, double mult);//改变数组

int main(void) {
    double dip[SIZE] = { 20.0, 17.66, 8.2, 15.3, 22.22 };
    printf("The original dip array:\n");
    show_array(dip, SIZE);
    mult_array(dip, SIZE, 2.5);
    printf("The dip array after calling mult_array():\n");
    show_array(dip, SIZE);
    return 0;
}
```

```
/* 显示数组的内容 */
void show_array(const double ar[], int n) {
        int i;
        for (i = 0; i < n; i++)
                printf("%8.3f ", ar[i]);
        putchar('\n');
}

/* 把数组的每个元素都乘以相同的值 */
void mult_array(double ar[], int n, double mult) {
        int i;
        for (i = 0; i < n; i++)
                ar[i] *= mult;
}
```



C:\Users\12096\Desktop\try.exe

```
The original dip array:
  20.000   17.660    8.200   15.300   22.220
The dip array after calling mult_array():
  50.000   44.150   20.500   38.250   55.550
```

const

假设已有：
```
double rates[5] = {88.99, 100.12, 59.45, 183.11, 340.5};
const double locked[4]={0.08,0.075,0.0725,0.07};
```

1.指针为const类型，表明不能通过指针修改其所向指的值。

```
const double *pd = rates;
*pd = 29.89;        //× 不允许
pd[2] = 222.22;     //×不允许
rates[0] = 99.99; // √允许，因为rates未被const限定
```

可以让pd指向别处

```
pd++;  /* 让pd指向rates[1] -- 没问题 */
```

2.把const数据或非const数据的地址初始化为指向const的指针或为其赋值是合法的

```
const double *pc = rates;  // √有效
pc = locked;      //√有效
pc = &rates[3]; //√有效
```

const

假设已有：
```
double rates[5] = {88.99, 100.12, 59.45, 183.11, 340.5};
const double locked[4]={0.08,0.075,0.0725,0.07};
```

3.可以声明并初始化一个不能指向别处的指针

```
double *const pc = rates; // pc指向数组的开始
pc = &rates[2]; // ×不允许，因为该指针不能指向别处
*pc = 92.99;     // √没问题 -- 更改rates[0]的值
```

4.在创建指针时还可以使用const两次，该指针既不能更改它所指向的地址，也不能修改指向地址上的值

```
double rates[5] = {88.99, 100.12, 59.45, 183.11, 340.5};
const double *const pc = rates;
// double double *const pc = rates; the same
pc = &rates[2]; //×不允许
*pc = 92.99;    //×不允许
```

Content

Memory management

C provides several functions for memory allocation and management.

function	Description
calloc(int num, int size)	Allocate an **initialized** array of **num** elements each with **size (in byte)**
malloc(int num)	Allocate an array of num bytes and leave them **uninitialized**
realloc(void *addr, int newsize)	Re-allocate memory at **addr**ess with **newsize**
free(void *addr)	Release a block of memory at **addr**ess

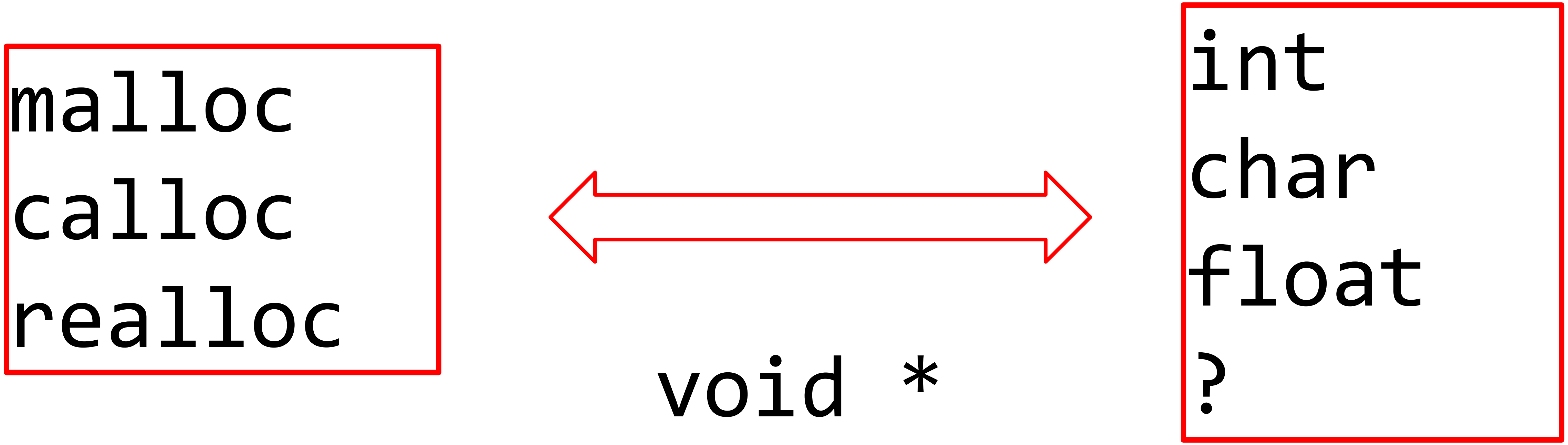
Memory management

You must use this library to use memory management function

⬇

```
#include <stdlib.h>
```

Memory management

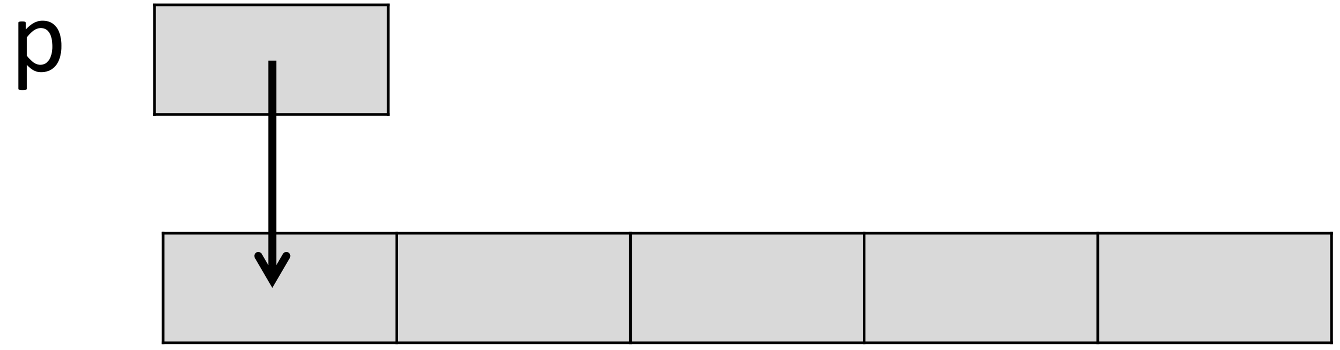


```
char *name;
name = calloc(100, sizeof(char));
```

取决于系统

Memory management

```
malloc
calloc
```



```
char *p;
p = (char*)calloc(100, sizeof(char));
```

calloc() function

Fixed array size, fixed memory

```
char name[100];
```

```
char *name;
name = (char*)calloc(100, sizeof(char));
```

Dynamic memory at address of name (100 bytes)

calloc() function

Fixed array size, fixed memory

```
int name[100];
```

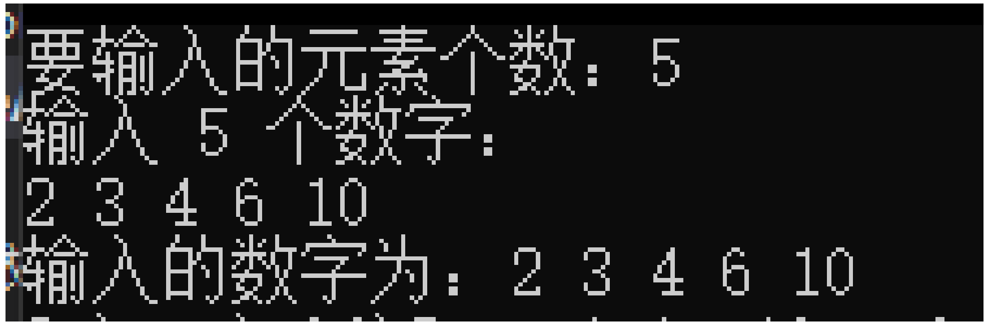
```
int *name;
name = (int*)calloc(100, sizeof(int));
```

How many bytes in total???

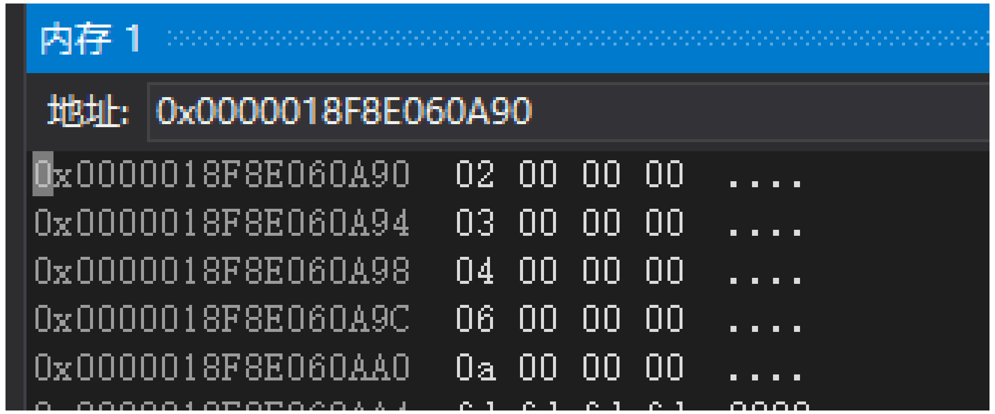
calloc() function

How to use calloc() to allocate memory for a pointer?

```
#include <stdio.h>
#include <stdlib.h>
main()
{
        int n;
        printf("要输入的元素个数: ");
        scanf("%d", &n);
        int* test_array = (int*)calloc(n, sizeof(int));
        printf("输入 %d 个数字: \n", n);
        for (int i = 0; i < n; i++){
                scanf("%d", &test_array[i]);
        }
        printf("输入的数字为: ");
        for (int i = 0; i < n; i++) {
                printf("%d ", test_array[i]);
        }
        free(test_array);
}
```



```
要输入的元素个数: 5
输入 5 个数字:
2 3 4 6 10
输入的数字为: 2 3 4 6 10
```



内存 1

地址: 0x0000018F8E060A90

```
0x0000018F8E060A90  02 00 00 00  ....
0x0000018F8E060A94  03 00 00 00  ....
0x0000018F8E060A98  04 00 00 00  ....
0x0000018F8E060A9C  06 00 00 00  ....
0x0000018F8E060AA0  0a 00 00 00  ....
```

malloc() function

Fixed array size, fixed memory

```
char name[100];
```

```
char *name;
name = (char*)malloc(100*sizeof(char));
```

Dynamic memory at address of name (100 bytes)

malloc() function

Fixed array size, fixed memory

```
float name[30];
```

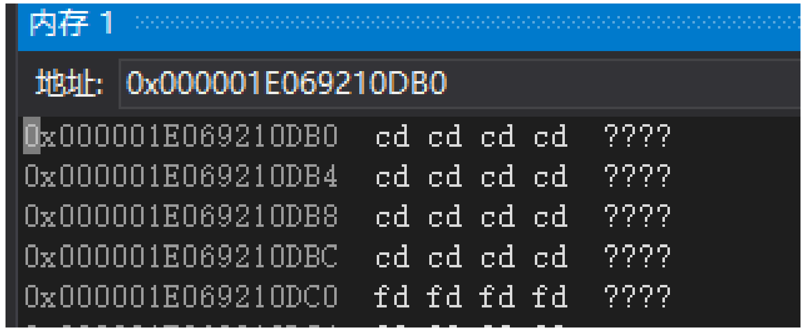
```
float *name;
name = (float*)malloc(30*sizeof(float));
```

How many bytes in total?

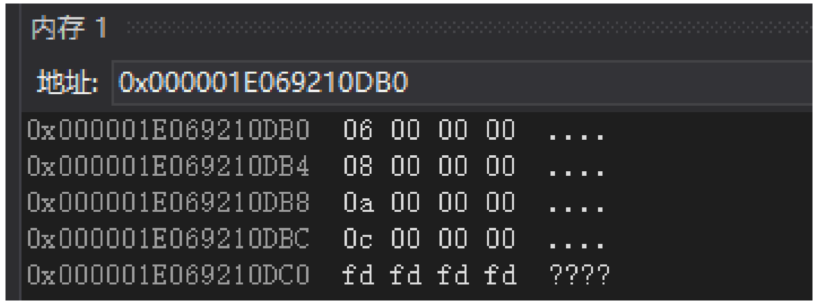
malloc() function

How to use malloc( ) to allocate memory for a pointer?

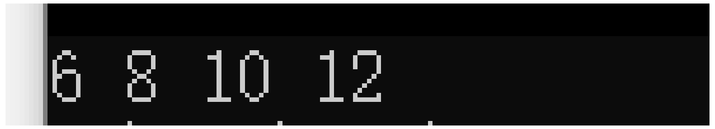
```
int* add_mats(int A[], int B[],int n)
{
    int* C = (int*)malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++)
        C[i] = B[i] + A[i];
    return C;
}

main(void)
{
    int A[4] = {1,2,3,4};
    int B[4] = {5,6,7,8};
    int *C = add_mats(A,B,4);
    for (int i = 0; i < 4 ; i++){
        printf("%d ",C[i]);
    }
}
```



```
0x000001E069210DB0  cd cd cd cd  ????
0x000001E069210DB4  cd cd cd cd  ????
0x000001E069210DB8  cd cd cd cd  ????
0x000001E069210DBC  cd cd cd cd  ????
0x000001E069210DC0  fd fd fd fd  ????
```



```
0x000001E069210DB0  06 00 00 00  ....
0x000001E069210DB4  08 00 00 00  ....
0x000001E069210DB8  0a 00 00 00  ....
0x000001E069210DBC  0c 00 00 00  ....
0x000001E069210DC0  fd fd fd fd  ????
```



```
6 8 10 12
```

calloc() & malloc()

```
char *name;

name = (char*)calloc(200, sizeof(char));

name = (char*)malloc(200*sizeof(char));
```

calloc() & malloc()

calloc()

contiguous/连续的
allocation

↓

allocates memory and
initializes all bits to zero

malloc()

memory
allocation

↓

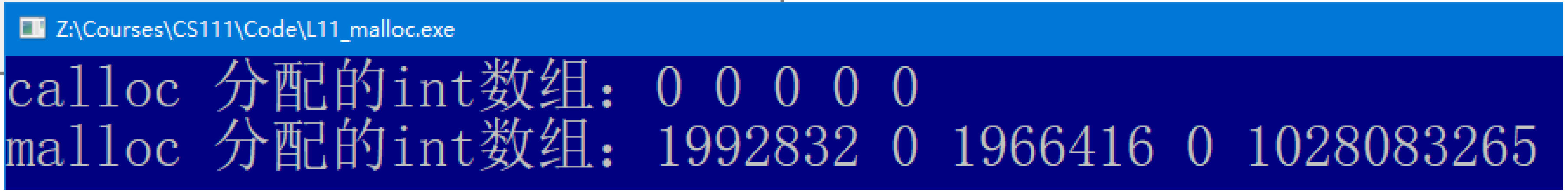
allocates memory and leaves
the memory uninitialized,
faster

Comparison on malloc() & calloc()

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    int n;
    printf("要输入的元素个数: ");
    scanf("%d", &n);
    int *test_array = (int*)calloc(n, sizeof(int));
    int *test_array1 = (int*)malloc(sizeof(int)*n);
    printf("calloc 分配的int数组: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", test_array[i]);
    }
    printf("\nmalloc 分配的int数组: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", test_array1[i]);
    }
}
```

calloc(): The space is initialized to zero.

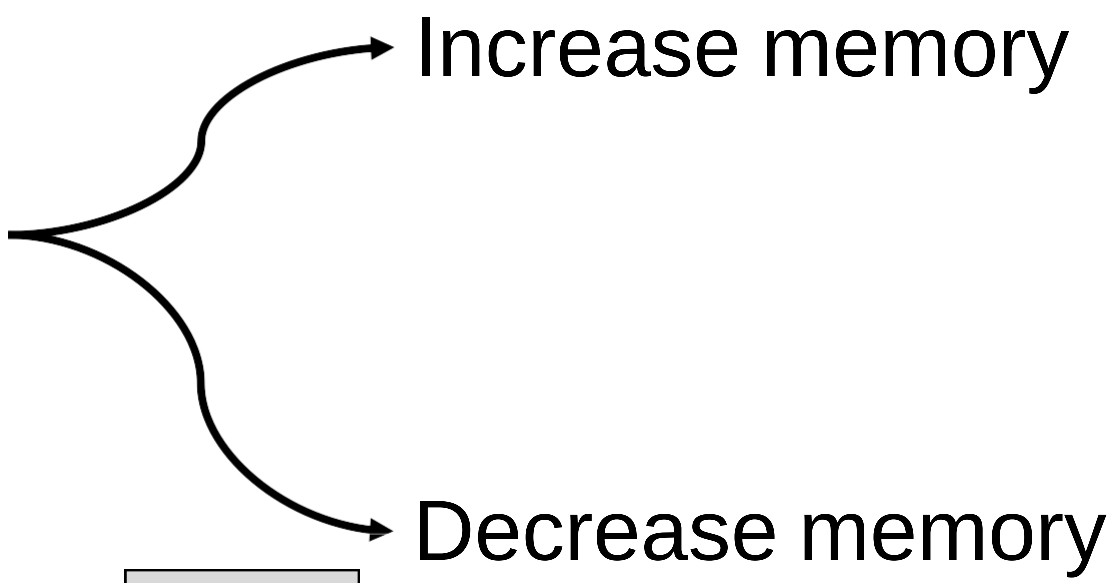
malloc(): The space is randomly initialized.


Z:\Courses\CS111\Code\L11_malloc.exe

```
calloc 分配的int数组: 0 0 0 0 0
malloc 分配的int数组: 1992832 0 1966416 0 1028083265
```

realloc() function

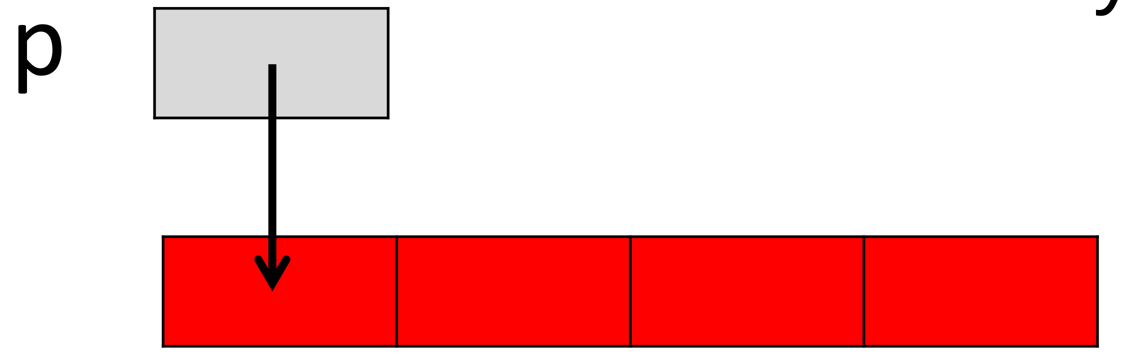
realloc()



Repeatedly allocation

↓

Re-allocates memory to the pointer variable



realloc() function

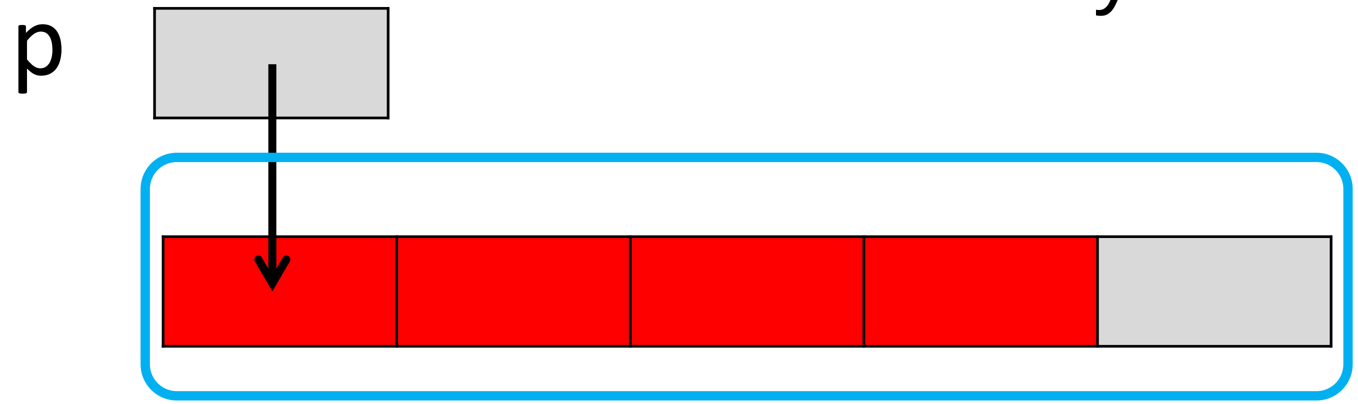
realloc()

Repeatedly allocation

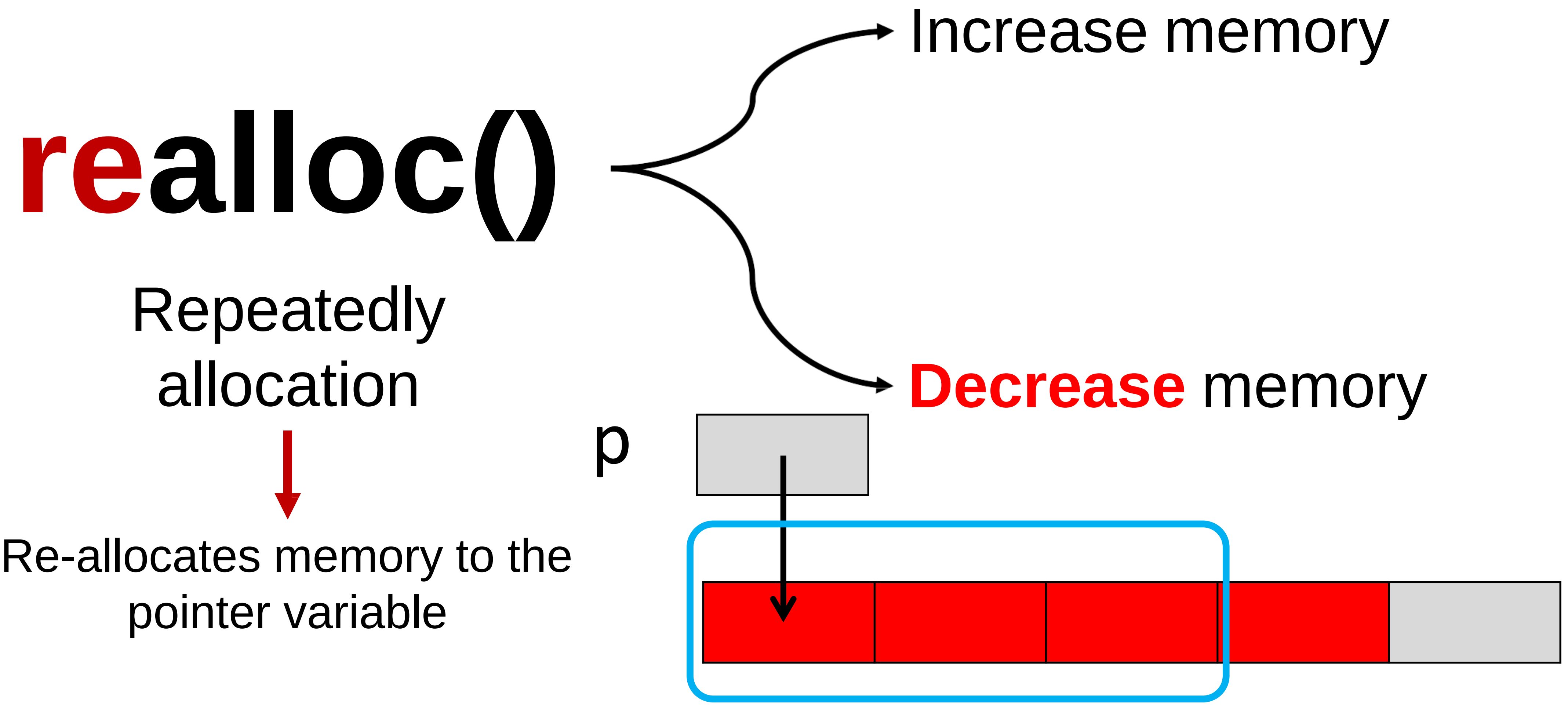
Re-allocates memory to the pointer variable

Increase memory

Decrease memory



realloc() function



Repeatedly allocation

Re-allocates memory to the pointer variable

realloc() function

Allocate memory at address of name (200 bytes)

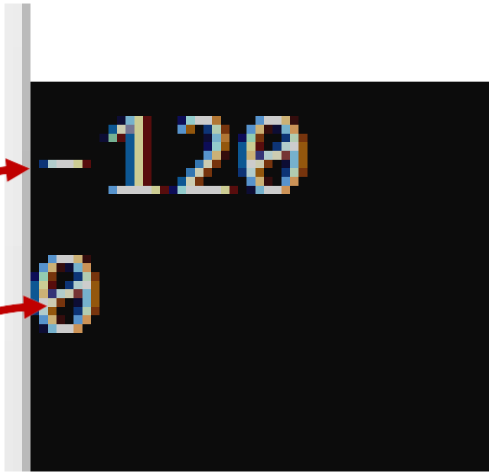
```
char *name;
name = (char*)malloc(200*sizeof(char));


name = (char*)realloc(name, 100*sizeof(char));
```

Resize the merry at address of name (100 bytes)

realloc() function

Why using realloc()?

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

int main(void)
{
    int *ptr;
    ptr = (int *)malloc(5 * sizeof(int));
    ptr[3] = -120;

    ptr = (int *)realloc(ptr, 10 * sizeof(int));
    printf("%d\n", ptr[3]);
    ptr = (int *)calloc( 5, sizeof(int));

    printf("%d", ptr[3]);

    return 0;
}
```



```
-120
0
```

If using malloc, the value will be erased!

null pointer

当分配内存失败时：

返回空指针(null pointer)

```
p=(int*) malloc(10000);
if(p==NULL){
/* allocation failed;
…
*/
}
```

```
if((p=(int*) malloc(10000))==NULL)
{/* allocation failed.*/
}
```

null pointer

当分配内存失败时:

返回空指针(null pointer)

```
if(p==NULL)
```

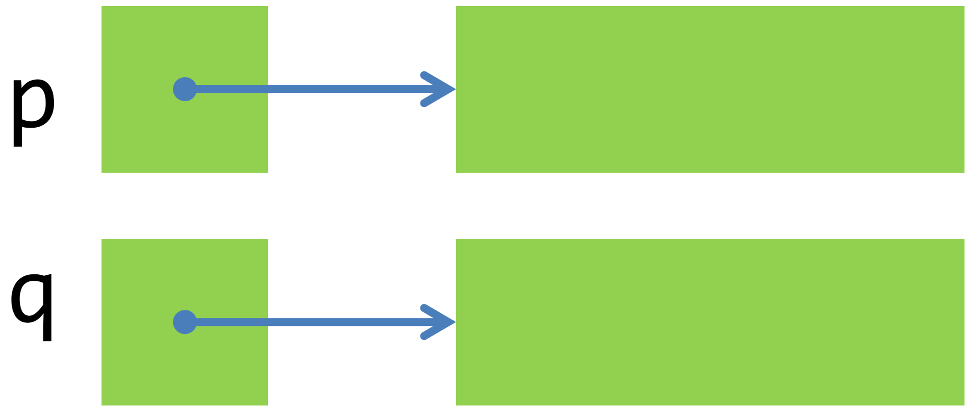
$\updownarrow$

```
if(!p)
```

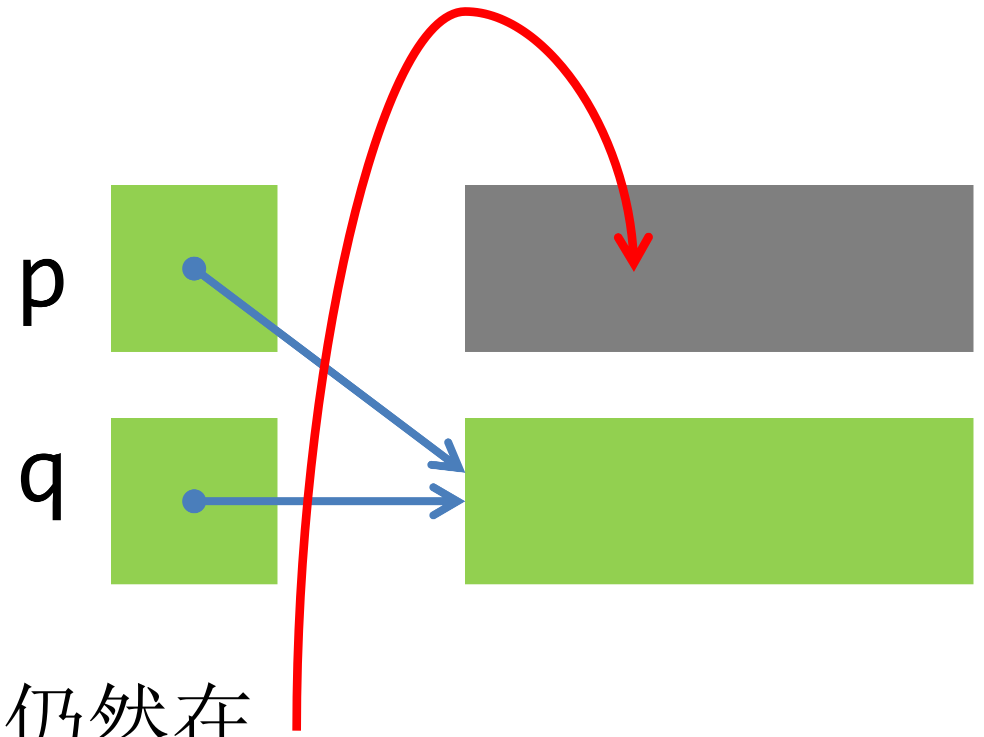
- 非空指针为真
- 空指针为假

free() function

```
p = (int*) malloc(…);
q = (int*) malloc(…);
```

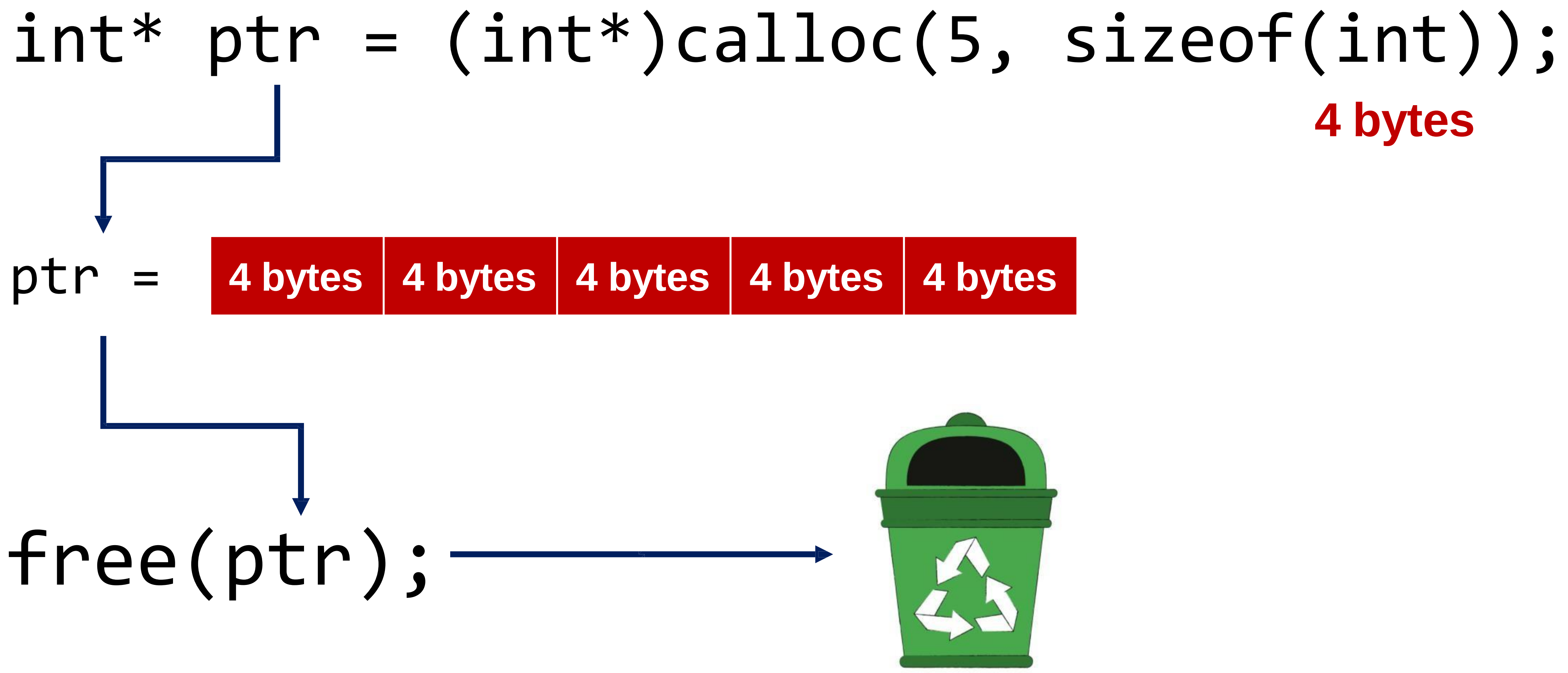


```
p = q;
```



此块内存不能再被使用，但是仍然在内存中存在，成为垃圾

free() function



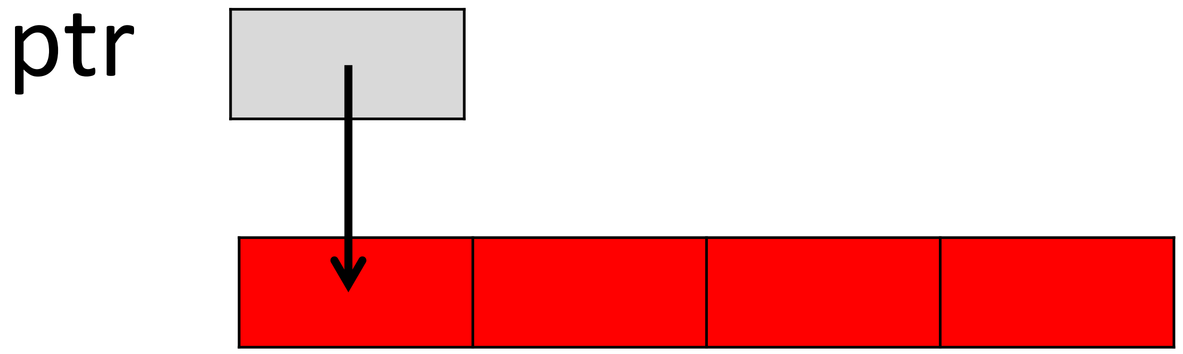
```c
int* ptr = (int*)calloc(5, sizeof(int));
```

```c
free(ptr);
```

free() function

```
int* ptr = (int*)calloc(5, sizeof(int));
free(ptr);
```

- free只是回收了指向的内容，指针还存，可以被重新指向。在被重新指向前禁止调用。



```
char* p = (char*)malloc(4);
free(p);
strcpy(p, “abc”);
```

❌

Content

1. User I/O
2. File I/O

Content

1. User I/O
2. File I/O

User I/O

I/O defines how machine reads human's input and put them on screen.

getchar
putchar

Only read/print a single char

gets
puts

Read/print a group of chars

scanf
printf

Read/print formatted values

getchar() and putchar()

- **getchar()** reads the next available single character and returns an integer representing the character in ASCII table.
- **putchar()** puts the given character on the screen.

```
int c = getchar();
putchar(c);
```

It reads and puts a single character!!!

gets() and puts()

- **gets()** reads a string (a group of characters) from user and puts it into a buffer (char[] or char*)
- **puts()** shows the string (a group of characters) on the screen

```
gets(char *s);
puts(char *s);
```

It reads and puts a group of characters!!!

scanf() and printf()

- **scanf()** scans the user input stream according to the provided format
- **printf()** prints the stream according to the format on the screen

```
scanf([formatted text], [arguments]);
printf([formatted text], [arguments]);
```

Formatted by specifiers

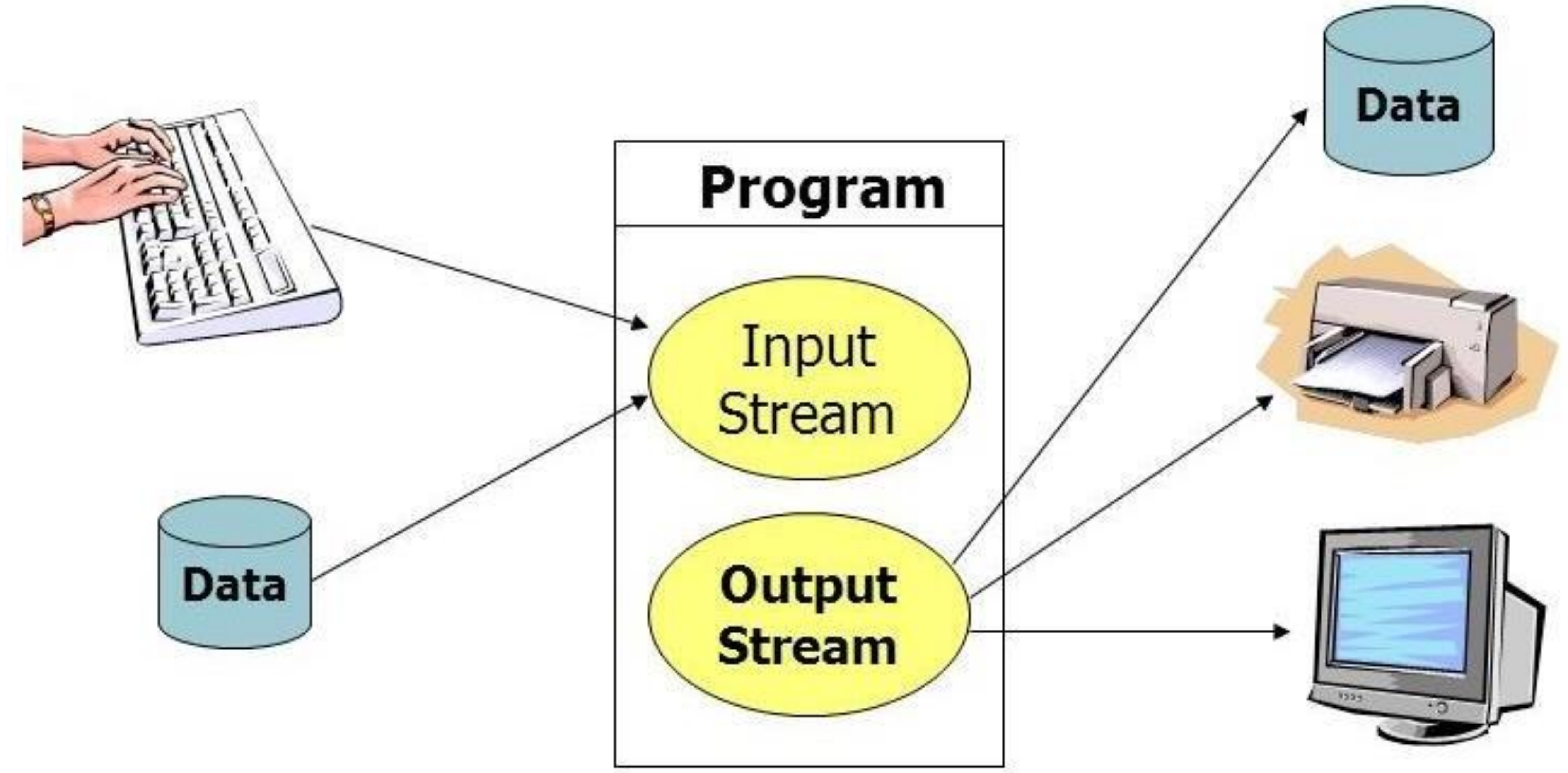
-	%d	int
-	%f	float
-	%c	char

f means formatted!!!

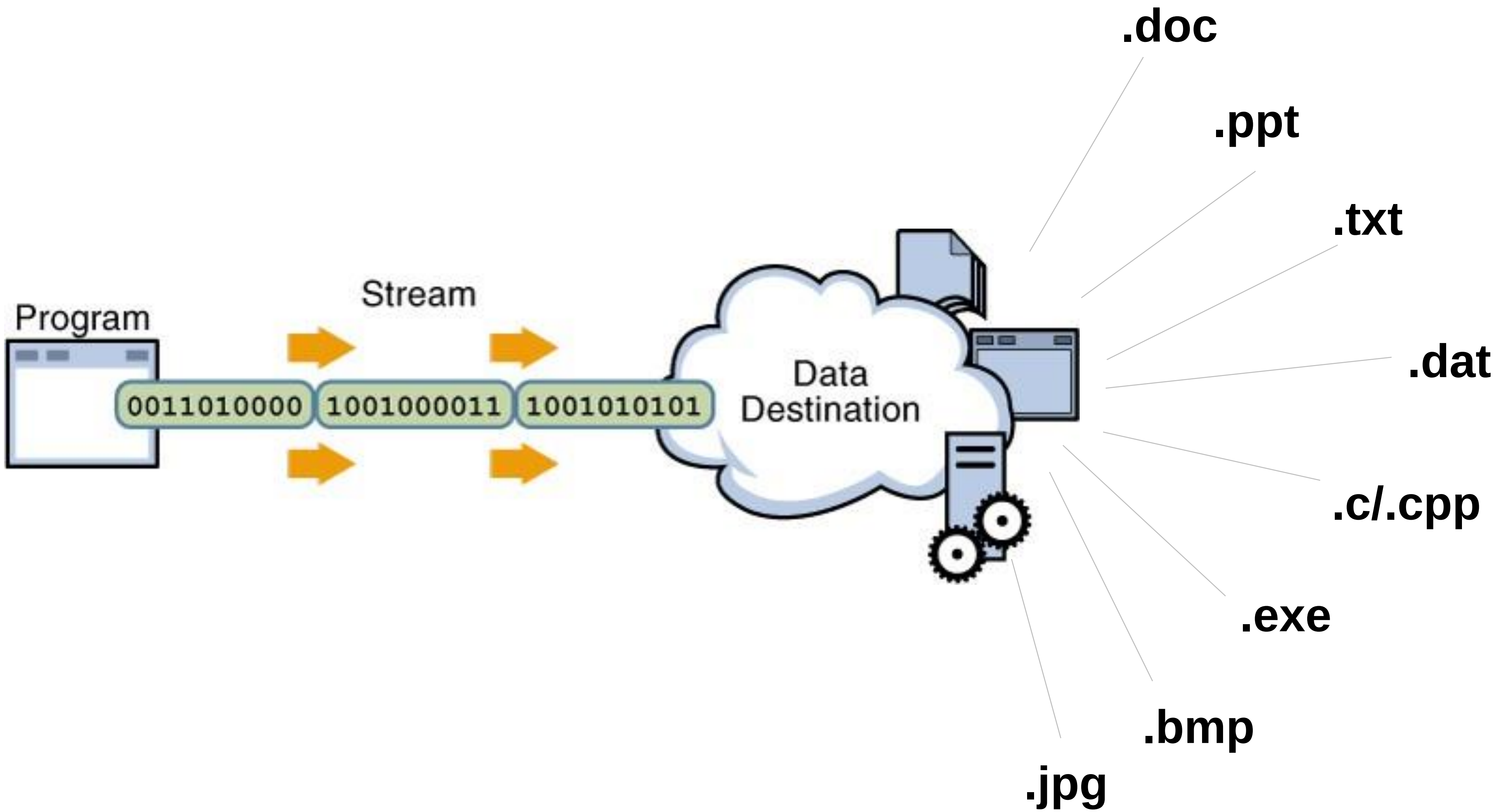
Content

1. User I/O
2. File I/O

File I/O in life



File I/O in life



Why file I/O is important?

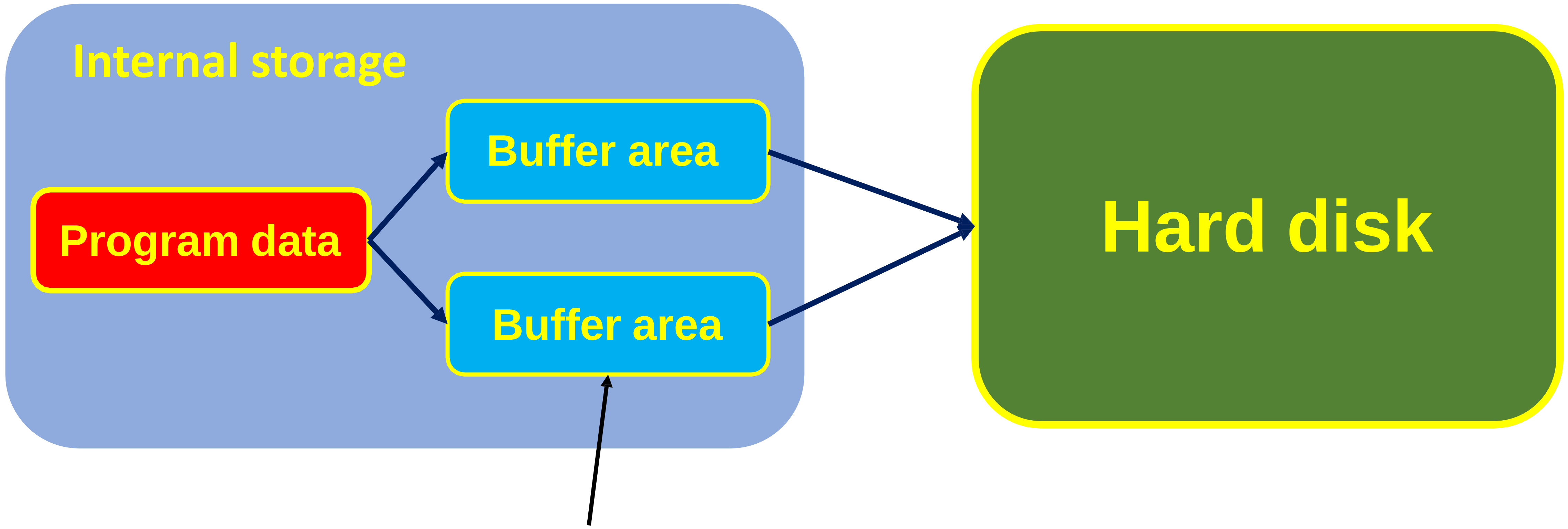
- You can save/preserve the data in file after terminating a program **(turn off PC will erase memory!!!**).

- You do not need to manually input the data but read from a file (**load a dataset**).

- You can transfer the data from one PC to another (**copy by hard drive or network**).

I/O buffering

- The system opens a file buffer in the memory.



A buffer is a memory area that stores data being transferred between two devices or between a device and an application.

File

文件的分类

- 从用户观点:

  特殊文件(标准输入输出文件或标准设备文件)。

  普通文件(磁盘文件)。

- 从操作系统的角度看，每一个与主机相连的输入输出设备看作是一个文件。

  例：输入文件：终端键盘

  输出文件：显示屏和打印机

- C语言对文件的处理方法：

缓冲文件系统：系统自动地在内存区为每一个正在使用的文件开辟一个缓冲区。用缓冲文件系统进行的输入输出又称为高级磁盘输入输出。

非缓冲文件系统：系统不自动开辟确定大小的缓冲区，而由程序为每个文件设定缓冲区。用非缓冲文件系统进行的输入输出又称为低级输入输出系统。

说明:
在UNIX系统下,用缓冲文件系统来处理文本文件,用非缓冲文件系统来处理二进制文件。ANSI C 标准只采用缓冲文件系统来处理文本文件和二进制文件。C语言中对文件的读写都是用库函数来实现。

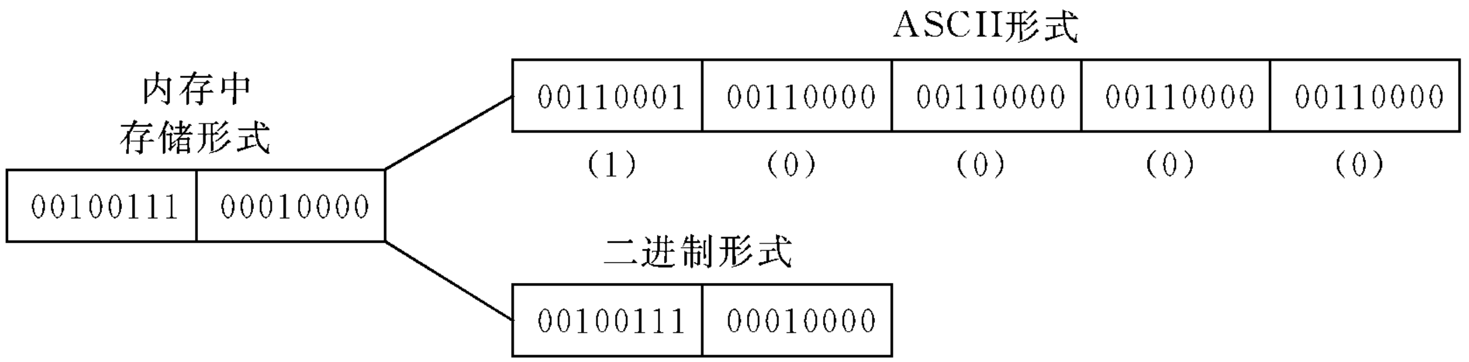
File

- 按数据的组织形式:

  ASCII文件(文本文件):每一个字节放一个ASCII代码

  二进制文件:把内存中的数据按其在内存中的存储形式原样输出到磁盘上存放。

  例：整数10000在内存中的存储形式以及分别按ASCII码形式和二进制形式输出如下图所示：



ASCII文件和二进制文件的比较：
ASCII文件便于对字符进行逐个处理，也便于输出字符。但一般占存储空间较多，而且要花费转换时间。
二进制文件可以节省外存空间和转换时间，但一个字节并不对应一个字符，不能直接输出字符形式。
一般中间结果数据需要暂时保存在外存上，以后又需要输入内存的，常用二进制文件保存。

File formats

ASCII file

binary file

.txt

Plain **text**
(data in characters)

data.txt

.csv

Comma **S**eparated **V**alues
(data structured by “,”)

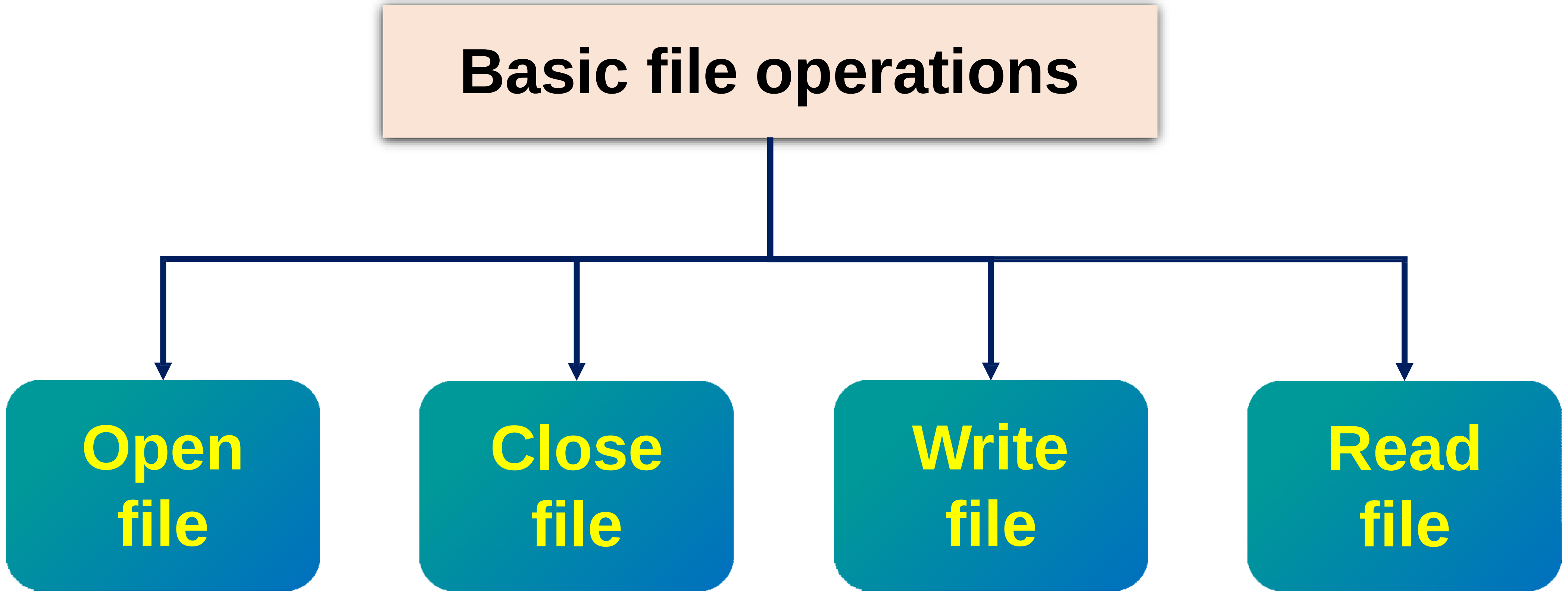
data.csv

.bin

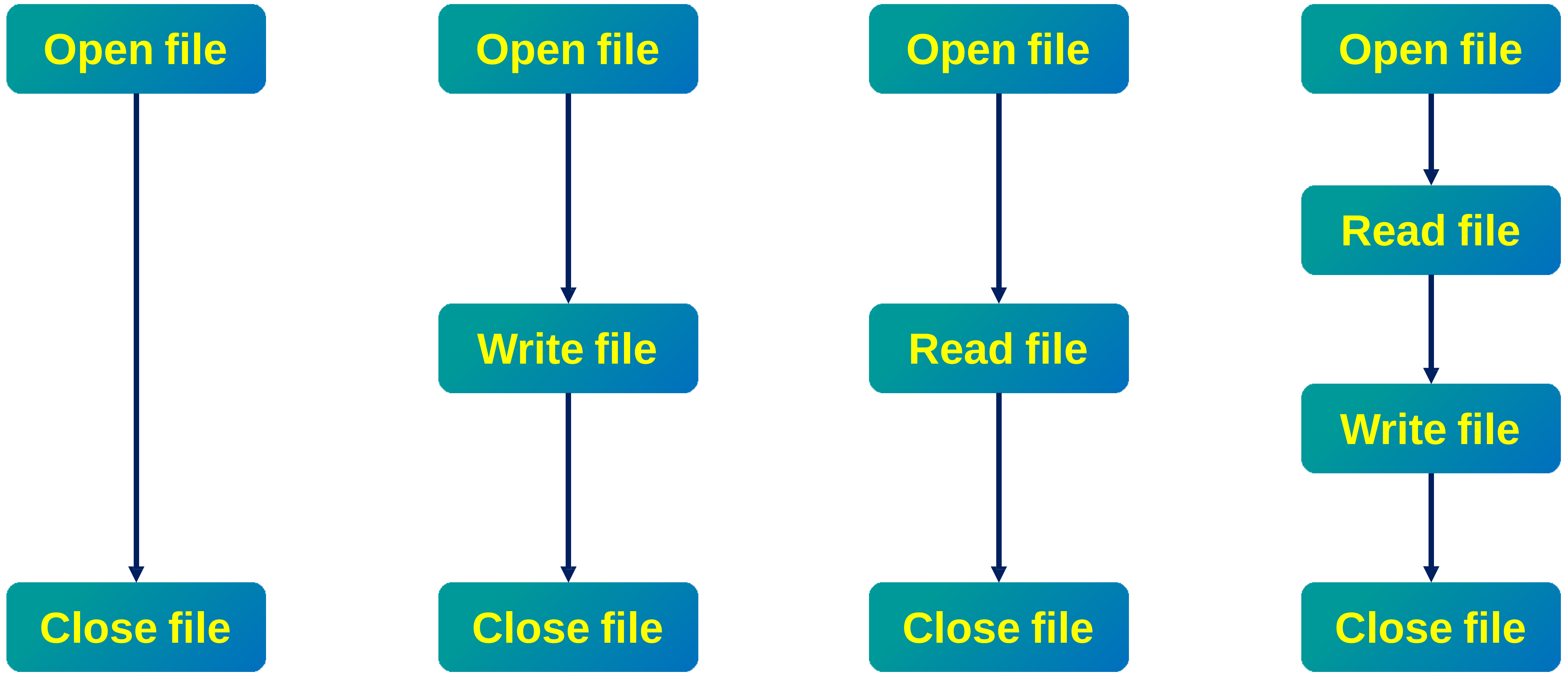
Binary values
(data in 0 and 1)

data.bin

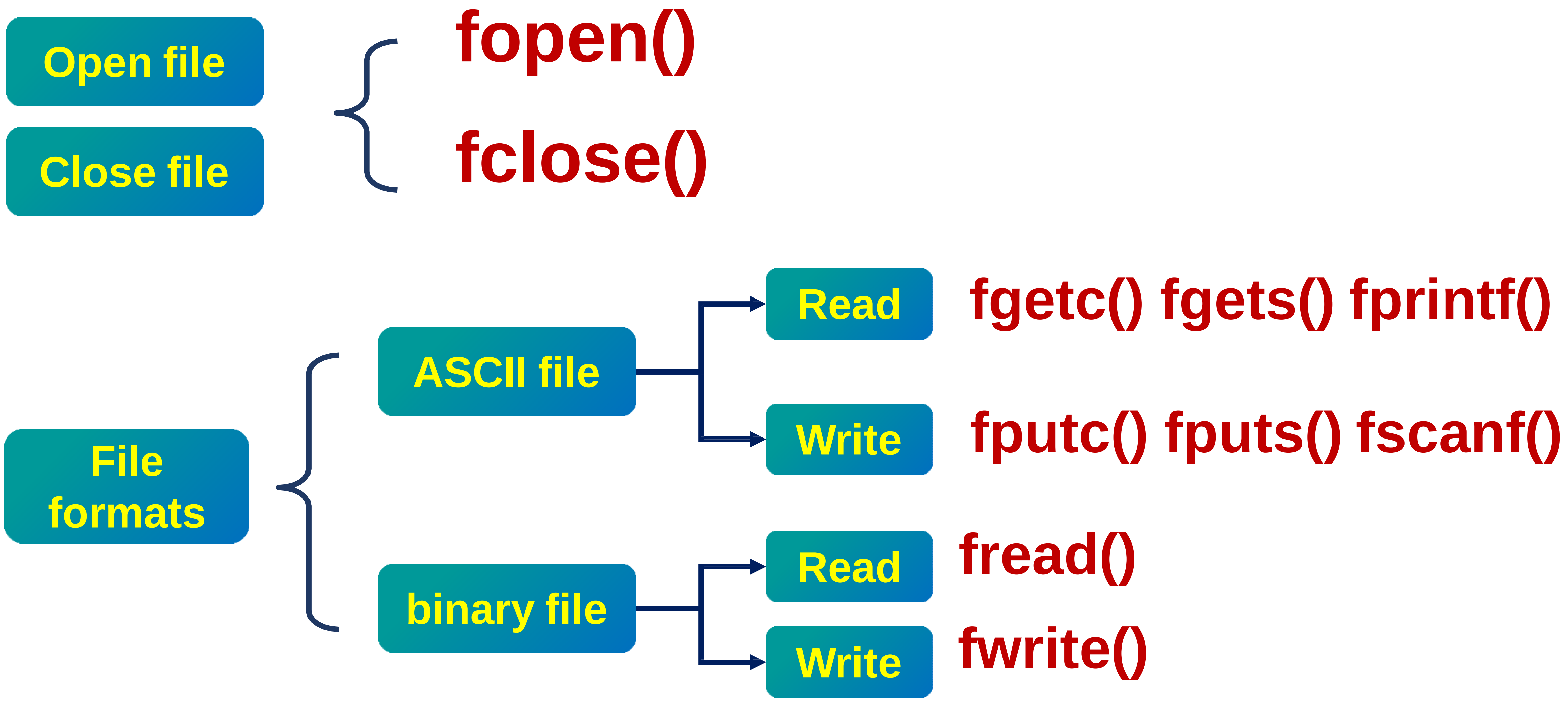
File I/O


Basic file operations
Open file
Close file
Write file
Read file

File I/O


Open file
Close file
Open file
Write file
Close file
Open file
Read file
Close file
Open file
Read file
Write file
Close file

Basic file operations



Open file

fopen() creates a new file or opens an existing file, it initializes an object of the type FILE

```
FILE *fopen( const char * filename,
const char * mode );
```

返回文件指针，
后续I/O操作目标。

Mode defines how you will interact with the file

Read **Write** **Append**

Open file

```
FILE *fopen(const char *filename, const char *mode);
```

Mode	Description
r	Opens an existing file for **r**eading
w	Opens a file for **w**riting, or create a new file if not exist
a	Opens an existing file for writing in **a**ppending mode
r+	Opens a file for reading and writing **both**
w+	Opens a file for reading and writing **both**
a+	Opens a file for reading and writing both, reading from start but writing in **a**ppending mode

Open file path

当前. c文件的路径下（相对路径）

```c
FILE* fp;

fp = fopen("test.txt", "w");


fp = fopen("C:\\Users\\Desktop\\c
files\\data.txt", "w");
```

系统的绝对路径

Open file Mode

To open ASCII files (txt, csv):

"r", "w", "a", "r+", "w+", "a+"

C11新增带x字母模式：”wx”,”wbx”,”w+x”,”wb+x”或”w+bx”。表明如果文件已存在或以独占模式打开文件，则打开文件失败

To open binary files:

"rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"

r = read

w = write

a = append

b = binary

如果如果使用任何一种**"w"**模式（不带x字母）打开一个现有文件，该文件的内容会被删除，以便程序在一个空白文件中开始操作。

Open a txt file

```c
#include <stdio.h>
int main(void)
{
    FILE* fp;

    fp = fopen("test.txt", "r");
    fp = fopen("test.txt", "w");
    fp = fopen("test.txt", "a");

    fp = fopen("test.txt", "r+");
    fp = fopen("test.txt", "w+");
    fp = fopen("test.txt", "a+");
}
```

Create a new file if not existing

Open a bin and csv file

bin file

```
#include <stdio.h>

main()
{
    FILE* fp;
    fp = fopen("test.bin","rb");
    fp = fopen("test.bin","wb");
    fp = fopen("test.bin","ab");

    fp = fopen("test.bin","r+b");
    fp = fopen("test.bin","w+b");
    fp = fopen("test.bin","a+b");
}
```

csv file

```
#include <stdio.h>

main()
{
    FILE* fp;
    fp = fopen("test.csv","rb");
    fp = fopen("test.csv","wb");
    fp = fopen("test.csv","ab");

    fp = fopen("test.csv","r+b");
    fp = fopen("test.csv","w+b");
    fp = fopen("test.csv","a+b");

}
```

Close file

fclose() closes a file, flushes data still pending in the buffer to the file, closes the file, and releases the used memory.

```
int fclose( FILE *fp );
```

如果成功关闭fclose()函数返回0，否则返回EOF

- ✓ Flushes data still pending in the buffer to file
- ✓ Closes the file
- ✓ Releases memory used for the file

Open and close a file

```
#include <stdio.h>
main()
{
    FILE* fp;

    fp = fopen("test.txt", "w+");

     // …

    fclose(fp);
}
```

Can be .txt, .bin, .csv

In a pair

1. No file, create a file
2. File exists, re-write the file

Write/Read a single char

Write a single character:

```
int fputc(int c, FILE *fp);
```

Read a single character:

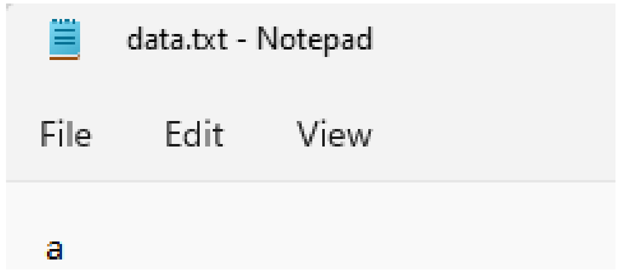
```
int fgetc(FILE *fp);
```

与putchar, getchar对应

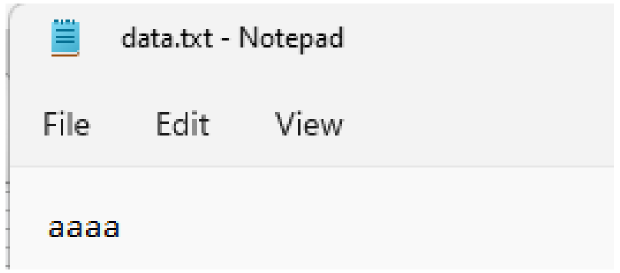
Write txt file by fputc()

```c
#include <stdio.h>
int main(void)
{
    FILE* fptr;
    fptr = fopen("data.txt","w+");
    char data = 'a';
    fputc(data, fptr);

    fclose(fptr);
}
```

```c
#include <stdio.h>
int main(void)
{
    FILE* fptr;
    fptr = fopen("data.txt","w+");
    char data = 'a';
    fputc(data, fptr);
    fputc(data, fptr);
    fputc(data, fptr);
    fputc(data, fptr);
    fclose(fptr);
}
```



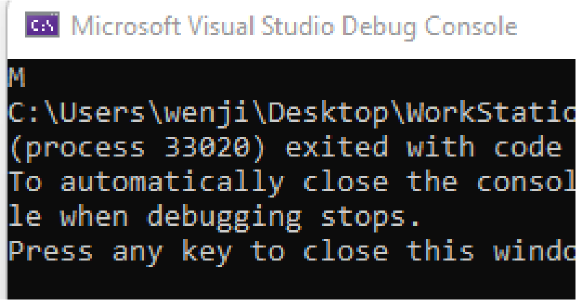
```
a
```



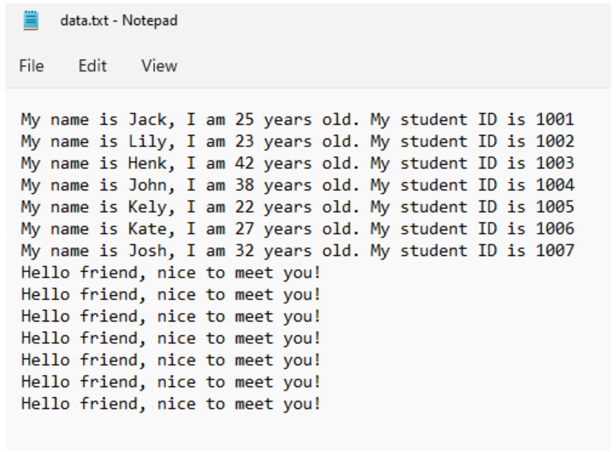
```
aaaa
```

Read txt file by fgetc()

```c
#include<stdio.h>

int main(void)
{
    FILE* fptr;
    fptr = fopen("data.txt","r");

    char c = fgetc(fptr);
    printf("%c", c);


    fclose(fptr);
}
```

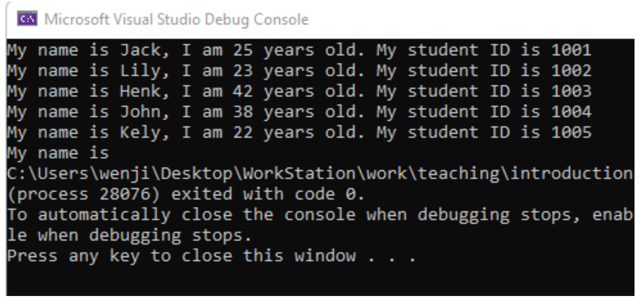


```
M
C:\Users\wenji\Desktop\WorkStatio
(process 33020) exited with code
To automatically close the consol
le when debugging stops.
Press any key to close this windo
```



```
My name is Jack, I am 25 years old. My student ID is 1001
My name is Lily, I am 23 years old. My student ID is 1002
My name is Henk, I am 42 years old. My student ID is 1003
My name is John, I am 38 years old. My student ID is 1004
My name is Kely, I am 22 years old. My student ID is 1005
My name is Kate, I am 27 years old. My student ID is 1006
My name is Josh, I am 32 years old. My student ID is 1007
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
```

```c
#include<stdio.h>

int main(void)
{
    FILE* fptr;
    fptr = fopen("data.txt","r");

    for (int i = 0; i < 300;i++)
    {
        char c = fgetc(fptr);
        printf("%c", c);
    }
    fclose(fptr);
}
```



```
My name is Jack, I am 25 years old. My student ID is 1001
My name is Lily, I am 23 years old. My student ID is 1002
My name is Henk, I am 42 years old. My student ID is 1003
My name is John, I am 38 years old. My student ID is 1004
My name is Kely, I am 22 years old. My student ID is 1005
My name is
C:\Users\wenji\Desktop\WorkStation\work\teaching\introduction
(process 28076) exited with code 0.
To automatically close the console when debugging stops, enab
le when debugging stops.
Press any key to close this window . . .
```

Write/Read a group of chars

Write a group of characters:

```
int fputs(const char *s, FILE *fp);
```

Read a group of characters:

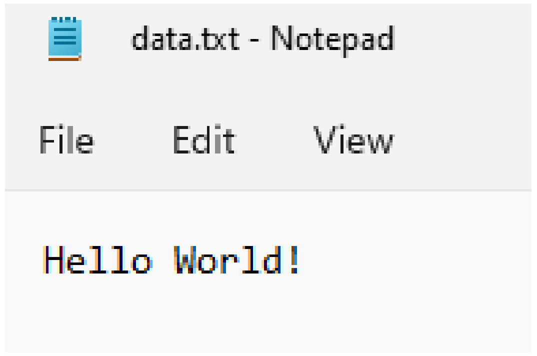
```
char* fgets(char *buf, int n, FILE *fp);
```

与puts, gets对应

Write txt file by fputs()

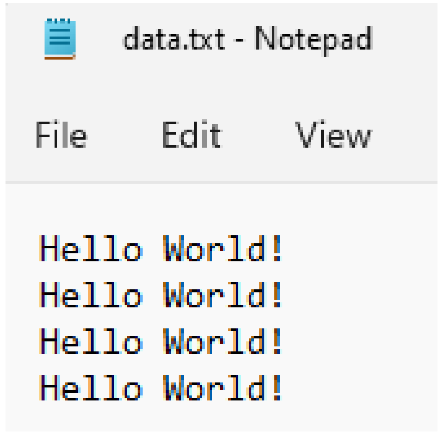
```c
#include<stdio.h>
int main(void)
{
    FILE* fptr;
    fptr = fopen("data.txt", "w");

    char data[] = "Hello World!";
    fputs(data, fptr);

    fclose(fptr);

}
```

```c
#include<stdio.h>
int main(void)
{
    FILE* fptr;
    fptr = fopen("data.txt", "w");
    char data[] = "Hello World!\n";
    fputs(data, fptr);
    fputs(data, fptr);
    fputs(data, fptr);
    fputs(data, fptr);
    fclose(fptr);
}
```



```
Hello World!
```

- puts指向stdout，在字符串末尾会添加\n
- fputs指向文件，不会自动添加\n

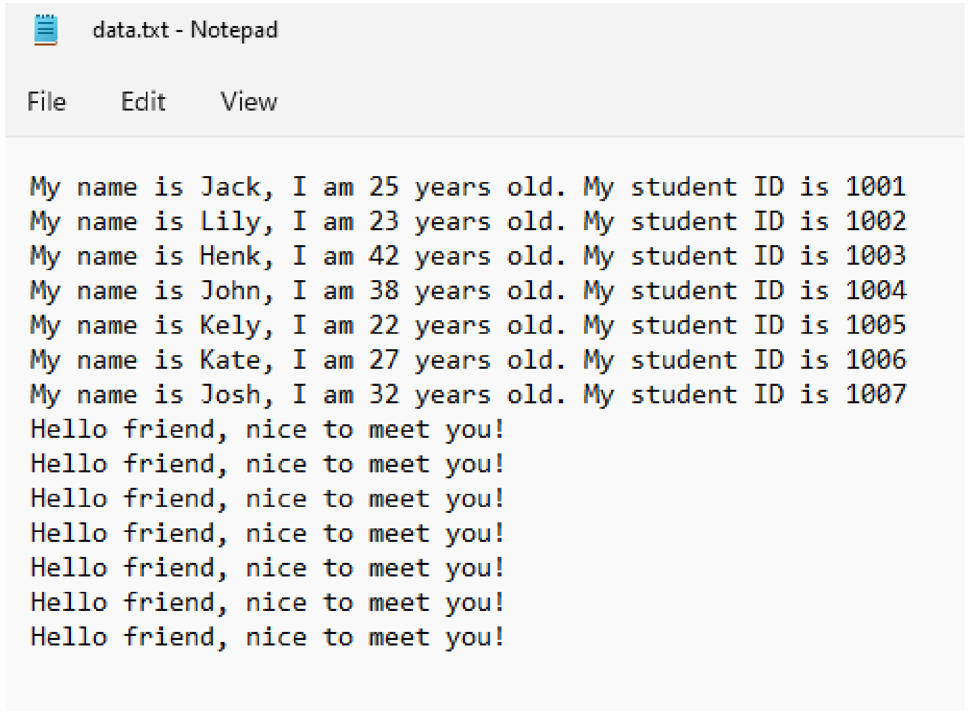


```
Hello World!
Hello World!
Hello World!
Hello World!
```

Read txt file by fgets()

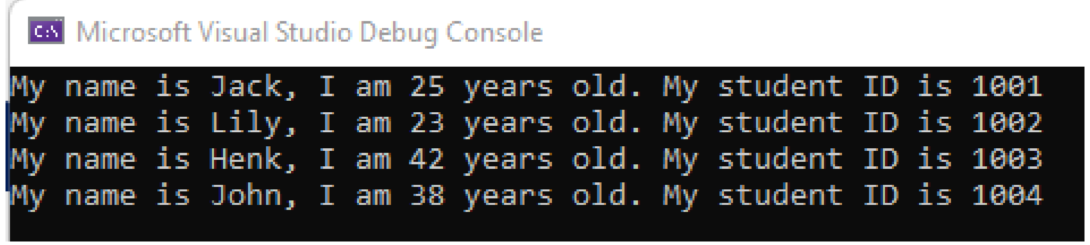
```
#include<stdio.h>

int main(void)
{
    FILE* fptr;
    fptr = fopen("data.txt", "r");

    char data[300];

    fgets(data, 100, fptr); printf("%s", data);
    fgets(data, 100, fptr); printf("%s", data);
    fgets(data, 100, fptr); printf("%s", data);
    fgets(data, 100, fptr); printf("%s", data);
    fclose(fptr);
}
```

Length of string (N-1, last is null)


data.txt - Notepad

File Edit View

```
My name is Jack, I am 25 years old. My student ID is 1001
My name is Lily, I am 23 years old. My student ID is 1002
My name is Henk, I am 42 years old. My student ID is 1003
My name is John, I am 38 years old. My student ID is 1004
My name is Kely, I am 22 years old. My student ID is 1005
My name is Kate, I am 27 years old. My student ID is 1006
My name is Josh, I am 32 years old. My student ID is 1007
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
```


Microsoft Visual Studio Debug Console

```
My name is Jack, I am 25 years old. My student ID is 1001
My name is Lily, I am 23 years old. My student ID is 1002
My name is Henk, I am 42 years old. My student ID is 1003
My name is John, I am 38 years old. My student ID is 1004
```

Write/Read formatted text

Write formatted characters:

```
fprintf(FILE *fp, const char *format, ...);
```

Read formatted characters:

```
fscanf(FILE *fp, const char *format, ...)
```

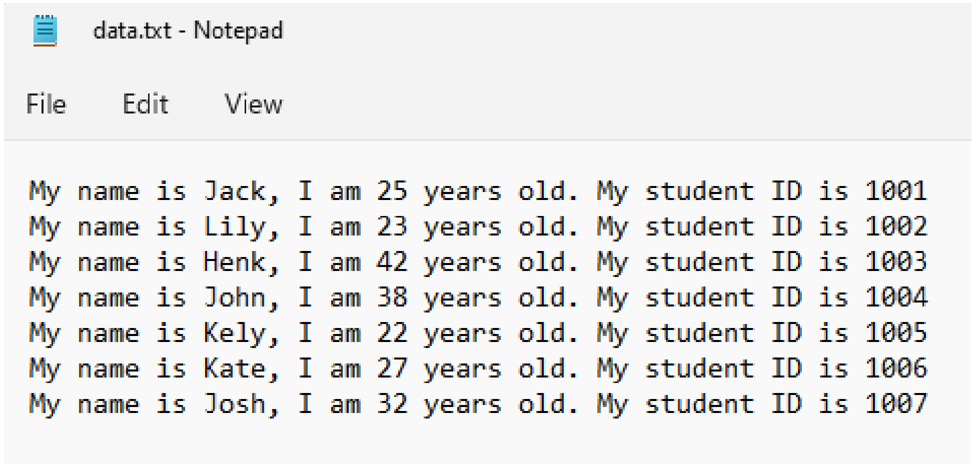
注意：
用fprintf和fscanf函数对磁盘文件读写，使用方便，容易理解，但由于在输入时要将ASCII码转换为二进制形式，在输出时又要将二进制形式转换成字符，花费时间比较多。因此，在内存与磁盘频繁交换数据的情况下，最好不用fprintf和fscanf函数，而用fread和fwrite函数。

与printf, scanf对应

Write txt file by fprintf()

```
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "w")

    char format[] = "My name is %s, I am %d years old. My student ID is %d\n";

    fprintf(fptr, format, "Jack", 25, 1001);
    fprintf(fptr, format, "Lily", 23, 1002);
    fprintf(fptr, format, "Henk", 42, 1003);
    fprintf(fptr, format, "John", 38, 1004);
    fprintf(fptr, format, "Kely", 22, 1005);
    fprintf(fptr, format, "Kate", 27, 1006);
    fprintf(fptr, format, "Josh", 32, 1007);

    fclose(fptr);
}
```

Writing mode

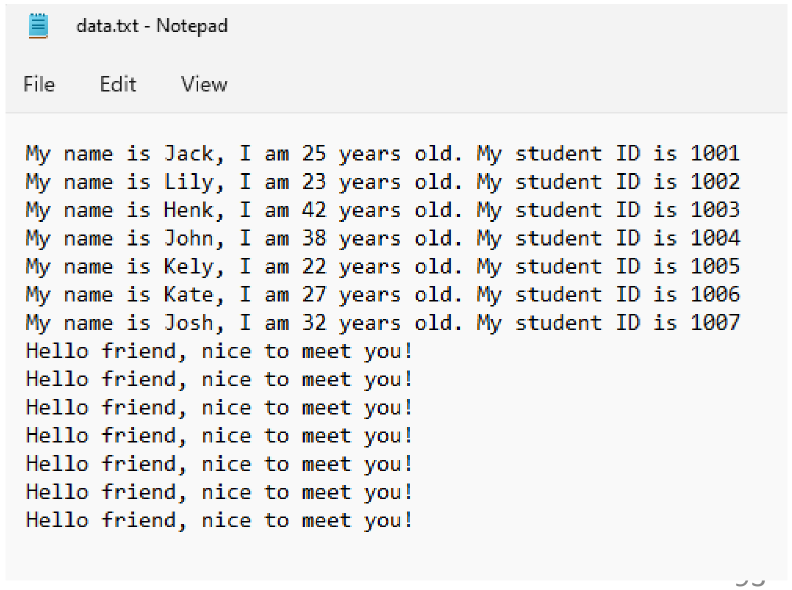


```
My name is Jack, I am 25 years old. My student ID is 1001
My name is Lily, I am 23 years old. My student ID is 1002
My name is Henk, I am 42 years old. My student ID is 1003
My name is John, I am 38 years old. My student ID is 1004
My name is Kely, I am 22 years old. My student ID is 1005
My name is Kate, I am 27 years old. My student ID is 1006
My name is Josh, I am 32 years old. My student ID is 1007
```

Write txt file by fprintf()

```c
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "a");

   char data[] = "Hello friend, nice to meet you!\n";

   fprintf(fptr, data);
   fprintf(fptr, data);
   fprintf(fptr, data);
   fprintf(fptr, data);
   fprintf(fptr, data);
   fprintf(fptr, data);
   fprintf(fptr, data);
    fclose(fptr);
}
```

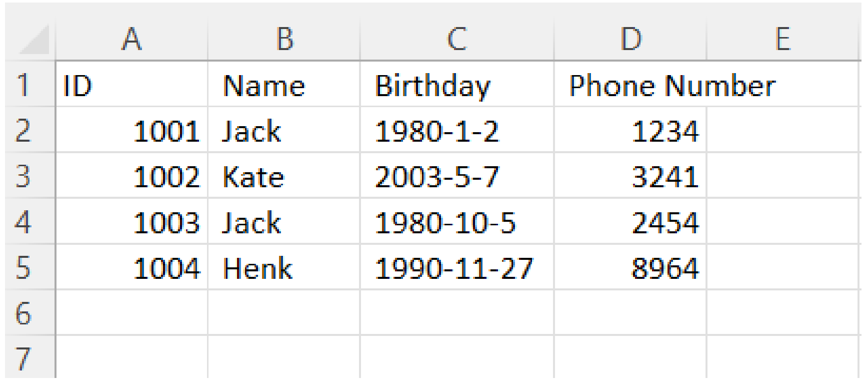
Appending mode (following last slide)



```
My name is Jack, I am 25 years old. My student ID is 1001
My name is Lily, I am 23 years old. My student ID is 1002
My name is Henk, I am 42 years old. My student ID is 1003
My name is John, I am 38 years old. My student ID is 1004
My name is Kely, I am 22 years old. My student ID is 1005
My name is Kate, I am 27 years old. My student ID is 1006
My name is Josh, I am 32 years old. My student ID is 1007
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
```

Write csv file by fprintf()

```
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.csv", "w");

   fprintf(fptr, "ID, Name, Birthday, Phone Number\n");
   fprintf(fptr, "%d, %s, %s, %d\n", 1001,"Jack","1980-1-2",1234);
   fprintf(fptr, "%d, %s, %s, %d\n", 1002,"Kate","2003-5-7",3241);
   fprintf(fptr, "%d, %s, %s, %d\n", 1003,"Jack","1980-10-5",2454);
   fprintf(fptr, "%d, %s, %s, %d\n", 1004,"Henk","1990-11-27",8964);


    fclose(fptr);
}
```

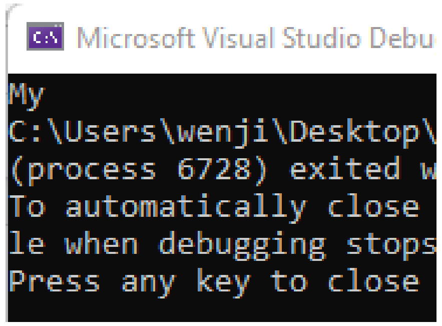


	A	B	C	D	E
1	ID	Name	Birthday	Phone Number	
2	1001	Jack	1980-1-2	1234	
3	1002	Kate	2003-5-7	3241	
4	1003	Jack	1980-10-5	2454	
5	1004	Henk	1990-11-27	8964	
6					
7					

Read txt file by fscanf()

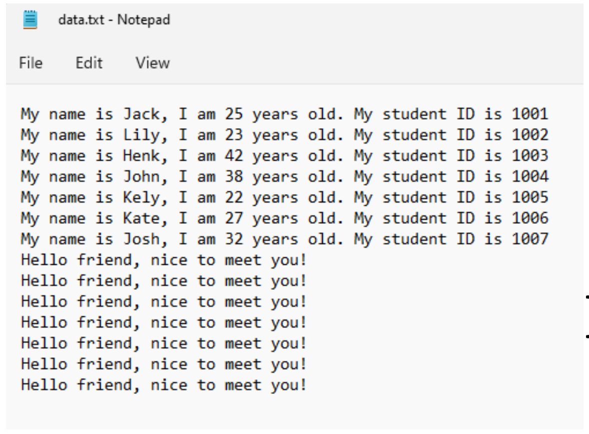
```c
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "r");

    char data[300];
    fscanf(fptr, "%s", data);

    printf("%s", data);

    fclose(fptr);
}
```

Word-basis reading!



Microsoft Visual Studio Debu

```
My
C:\Users\wenji\Desktop\
(process 6728) exited w
To automatically close
le when debugging stops
Press any key to close
```

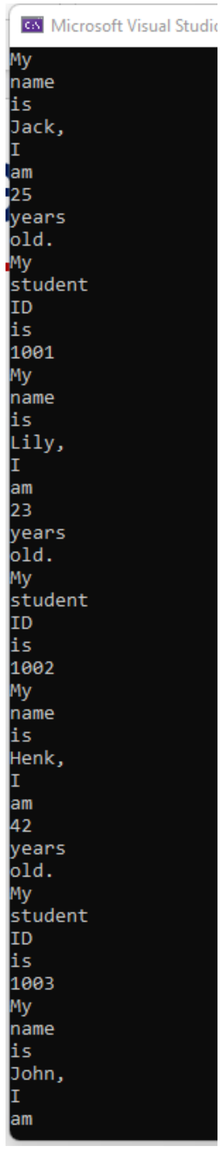


data.txt - Notepad

File Edit View

```
My name is Jack, I am 25 years old. My student ID is 1001
My name is Lily, I am 23 years old. My student ID is 1002
My name is Henk, I am 42 years old. My student ID is 1003
My name is John, I am 38 years old. My student ID is 1004
My name is Kely, I am 22 years old. My student ID is 1005
My name is Kate, I am 27 years old. My student ID is 1006
My name is Josh, I am 32 years old. My student ID is 1007
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
Hello friend, nice to meet you!
```

```c
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data.txt", "r");

    char data[300];
    for(int i = 0; i < 100; i ++)
    {
        fscanf(fptr, "%s", data);
        printf("%s\n", data);
    }

    fclose(fptr);
}
```



Microsoft Visual Studi

```
My
name
is
Jack,
I
am
25
years
old.
My
student
ID
is
1001
My
name
is
Lily,
I
am
23
years
old.
My
student
ID
is
1002
My
name
is
Henk,
I
am
42
years
old.
My
student
ID
is
1003
My
name
is
John,
I
am
```

Read and write

putchar	fputc
getchar	fgetc
puts	fputs
gets	fgets
printf	fprintf
scanf	fscanf
只能通过标准输入、标准输出、错误输出进行输入/输出	通过指定设备进行输入/输出 (stdin, stdout, stderr)

bin file

Write binary file:

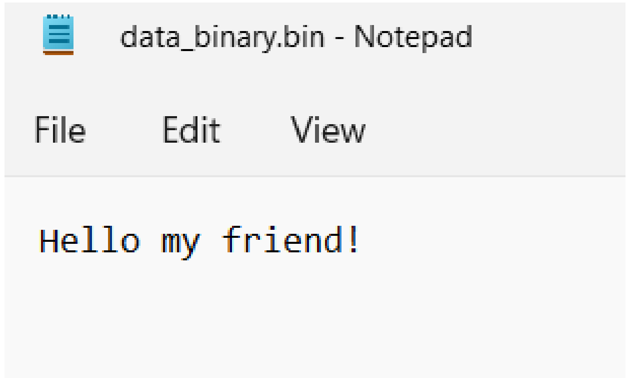
```
fwrite(const void *ptr, int size_of_elements, int
number_of_elements, FILE *fp);
```

Read binary format:

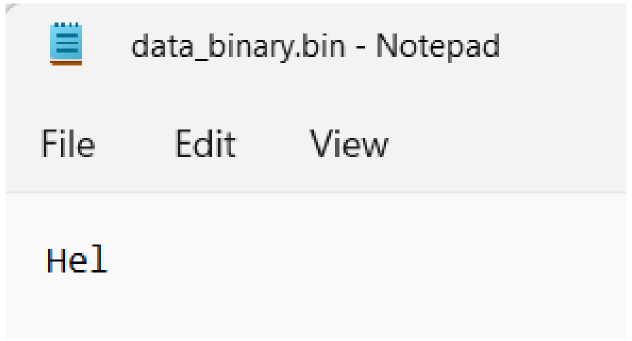
```
int fread(void *ptr, int size_of_elements, int
number_of_elements, FILE *fp);
```

Write bin file by fwrite()

```c
#include<stdio.h>

main()
{
    FILE* fptr;
    fptr = fopen("data_binary.bin", "wb");

    char data[] = "Hello my friend!\n";

    fwrite(data, sizeof(data), 1, fptr);

    fwrite(data, sizeof(char), sizeof(data), fptr);

    fwrite(data, sizeof(char), 3, fptr);

    fclose(fptr);
}
```



```
Hello my friend!
```



```
Hel
```

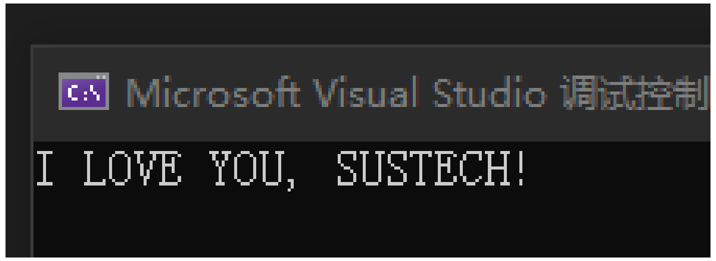
Case study

Read the file, convert lowercase to uppercase, and write to the file

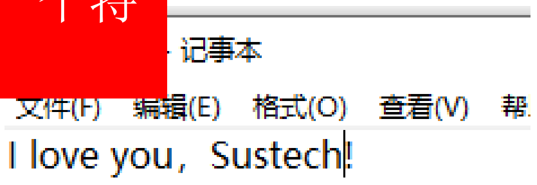
```c
int main()
{
    FILE* fp,*fp_output;
    char str[100];
    char ch;
    if ((fp = fopen("test.txt", "r")) == NULL){
        printf("file cannot open!");
        exit(0);}
    fp_output = fopen("output.txt", "w+");
    while ((ch = fgetc(fp)) != EOF) {
        if (ch >= 'a' && ch <= 'z') ch -= 32;
        fputc(ch, fp_output);}
    fclose(fp);
    fclose(fp_output);
    printfile("output.txt");
    return 0;
}
```

```c
void printfile(char* name)
{
    FILE* fp = fopen(name, "r");
    char ch;
    while ((ch = fgetc(fp)) != EOF)
        printf("%c", ch);
    fclose(fp);
}
```

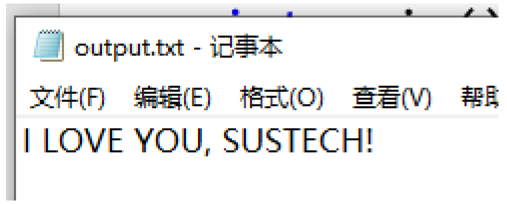
fgetc()读到文件结尾时将返回一个特殊值EOF



```
I LOVE YOU, SUSTECH!
```



```
I love you, Sustech!
```



```
I LOVE YOU, SUSTECH!
```

Random Access

Treat a file like an array and move directly to any particular byte in a file opened by fopen():

```
fseek(FILE *fp, offset, mode);
```

returns the current position in a file as a long value:

```
ftell(FILE *fp)
```

fseek & ftell

```
fseek(FILE *fp, offset, mode);
```

offset(偏移量)：示从起始点开始要移动的距离。该参数必须是一个long类型的值，可以为正（前移）、负（后移）或0（保持不动）。

mode(模式)：确定起始点。

如果一切正常，fseek()的返回值为0；如果出现错误（如试图移动的距离超出文件的范围），其返回值为-1。

Mode	Measures Offset From
SEEK_SET	Beginning of file
SEEK_CUR	Current position
SEEK_END	End of file

fseek & ftell

```
ftell(FILE *fp)
```

ftell()函数的返回类型是long，返回的是参数指向文件的当前位置距文件开始处的字节数

```
fseek(fp, 0L, SEEK_END);            //当前位置设置在文件结尾
last = ftell(fp);        //从文件开始到文件结尾处的字节数赋给last
for (count = 1L; count <= last; count++)
{
    fseek(fp, -count, SEEK_END); /* go backward */
    ch = getc(fp);
    printf("%c", ch);
}
```

Error detection

- int feof(FILE *fp);

  作用：来判断文件是否真的结束。如果是文件结束，函数feof（fp）的值为 1（真）；否则为 0 （假）。

- int ferror(FILE *fp)；

  作用：返回0，表示未出错；返回非0，表示出错。

- void clearerr(FILE *fp)

  作用：使文件错误标志和文件结束标志置为0。只要出现错误标志，就一直保留，直到对同一文件调用clearerr函数或rewind函数，或任何其他一个输入输出函数。