C程序设计基础

Introduction to C programming

Lecture 13:self-defined types

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 L12 I/O

Pointer and function

Memory management(advanced uses)

User I/O

File I/O

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!!!

- 数组的操作

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。

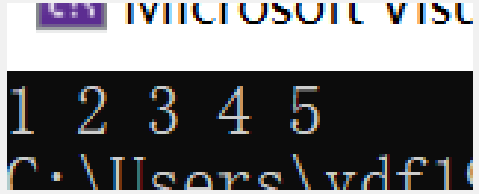
一般定义形式为：

类型名 *函数名（参数列表）

Function can return pointer

```
int * myFunction()
{

 . . .

}
```

```c
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;
}
```



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**。

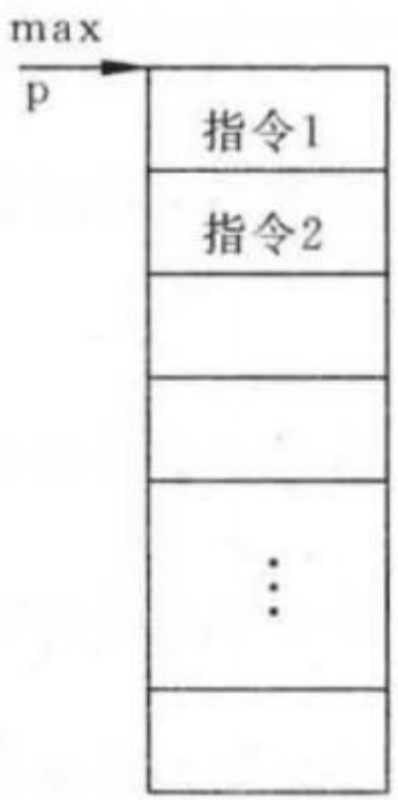
```
int (*p)(int, int);
```

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）！！

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;
}
```

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
```

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

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

Memory management

```
malloc
calloc
realloc
```

```
void *
```

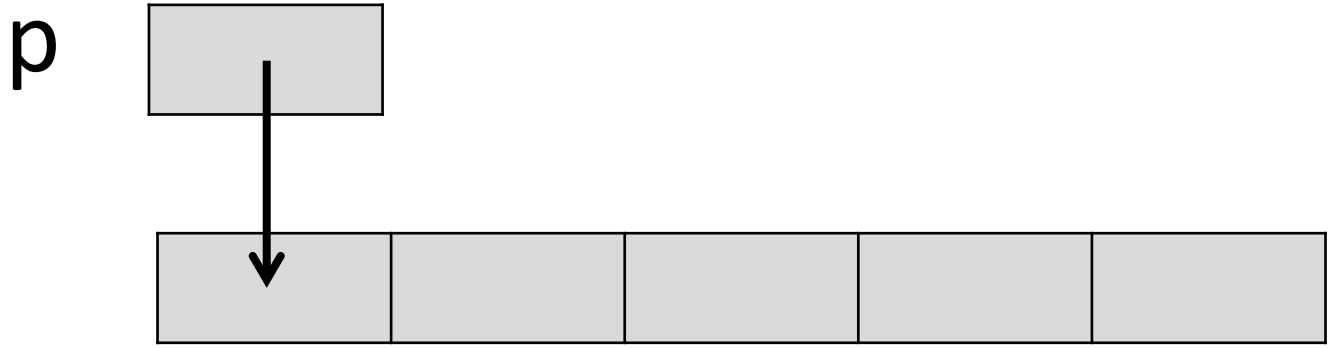
```
int
char
float
?
```

```
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() & 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

realloc() function

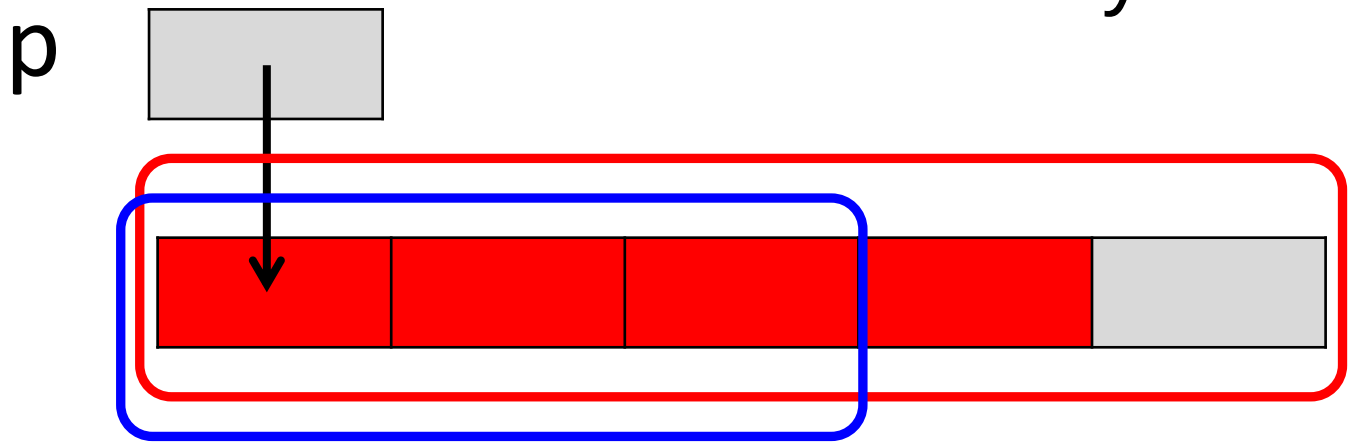
realloc()

Repeatedly allocation

Re-allocates memory to the pointer variable

Increase memory

Decrease memory



null pointer

当分配内存失败时：

返回空指针(null pointer)

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

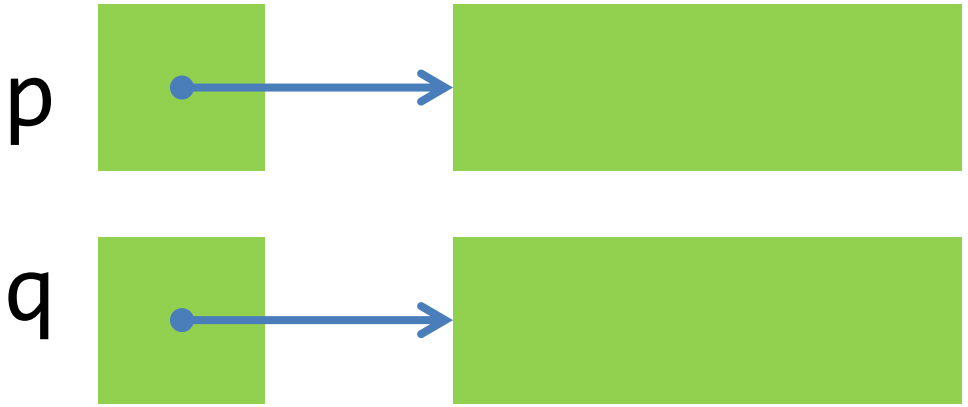
↕

```
if(!p)
```

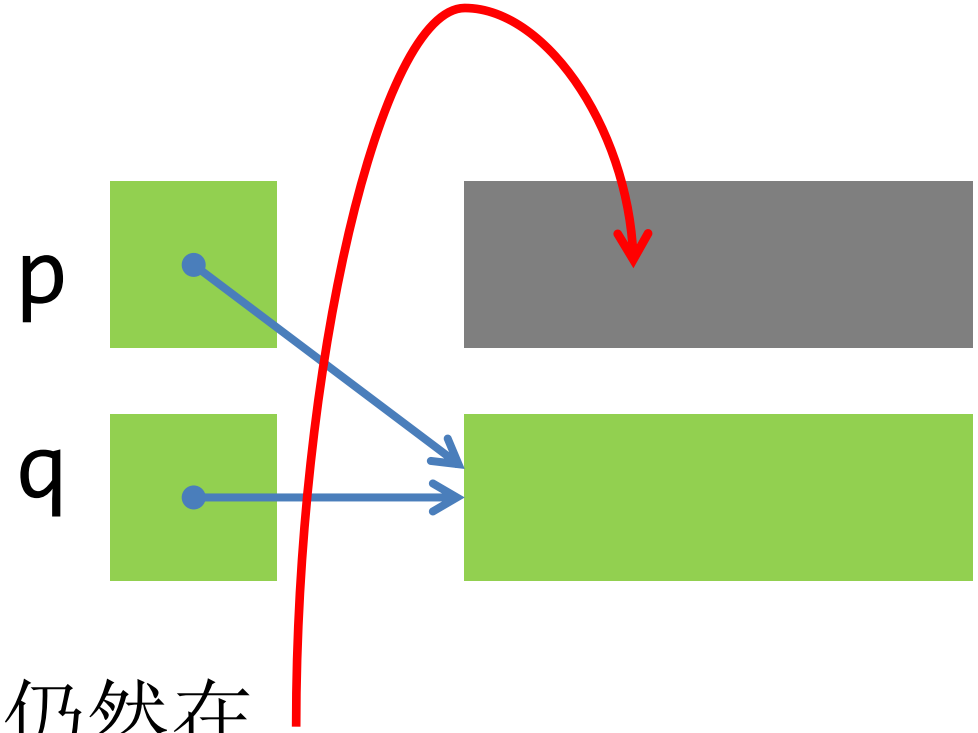
- 非空指针为真
- 空指针为假

free() function

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




```
p = q;
```

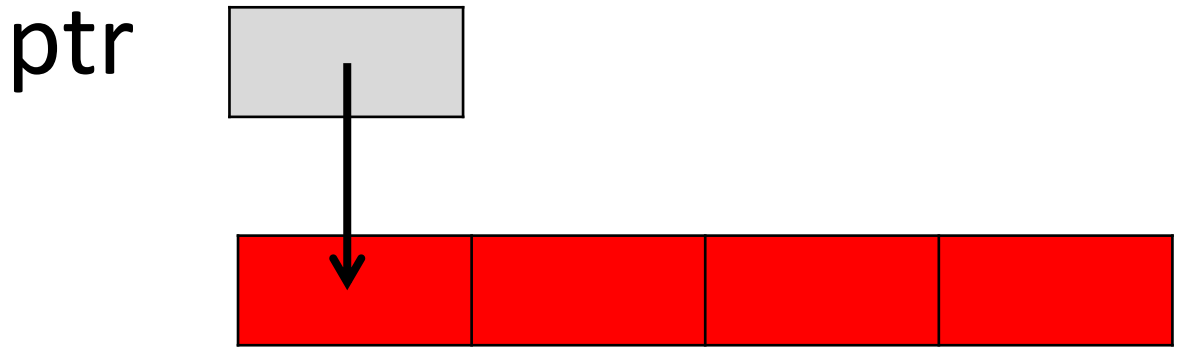


此块内存不能再被使用，但是仍然在内存中存在，成为垃圾

free() function

```
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

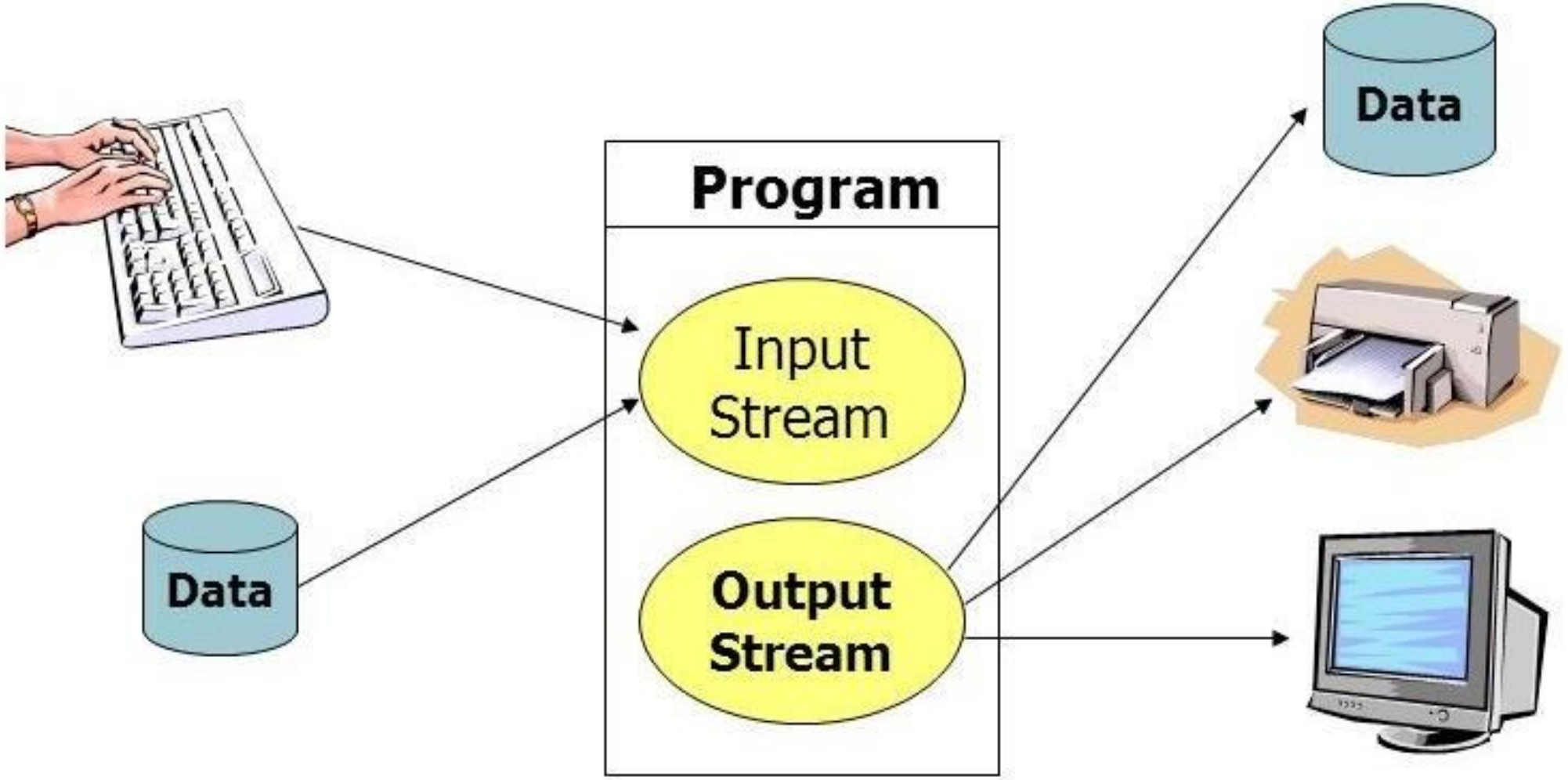
1. User I/O
2. File I/O

User I/O

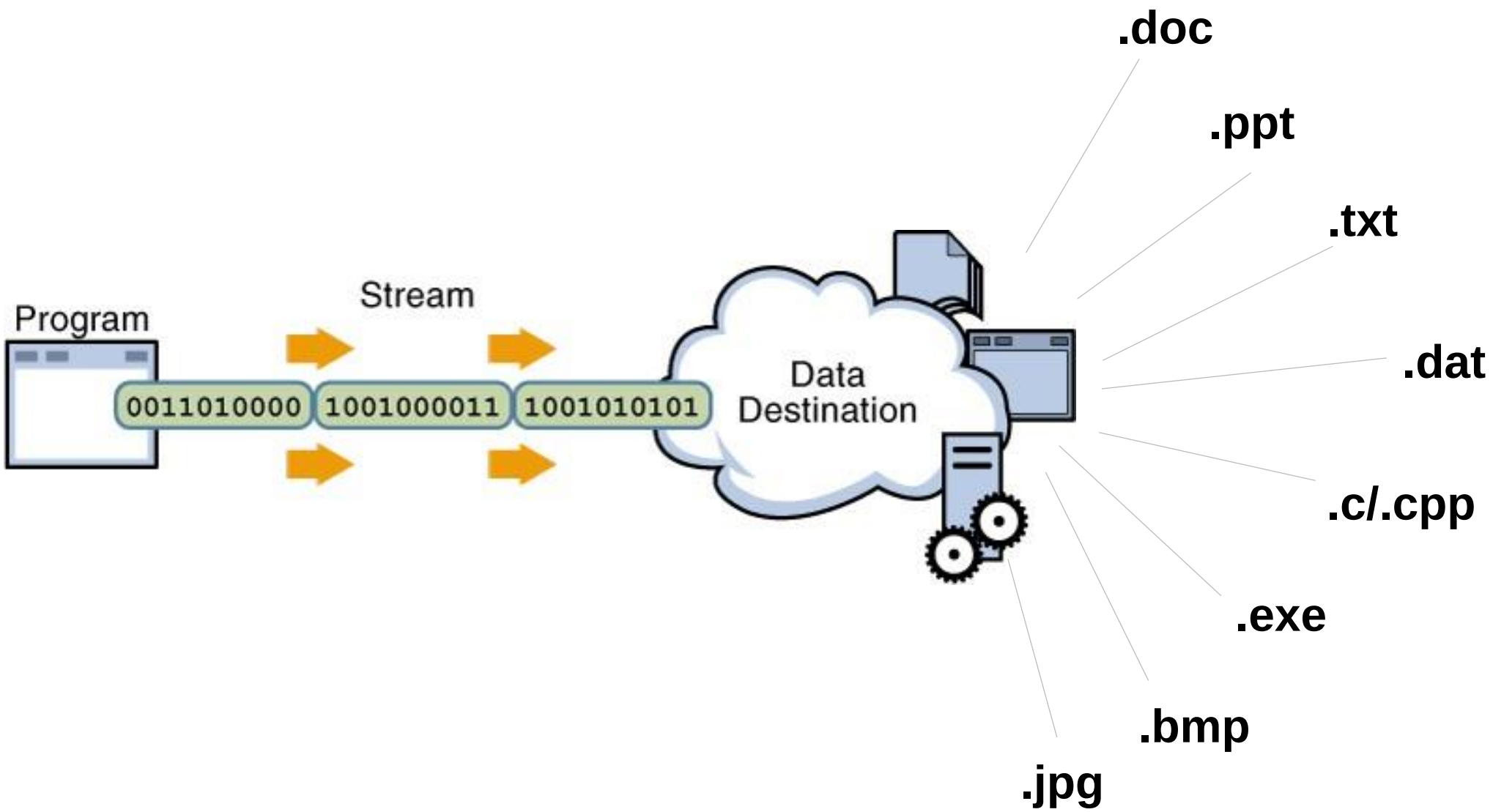
I/O defines how machine reads human's input and put them on screen.

getchar putchar	gets puts	scanf printf
Only read/print a single char	Read/print a group of chars	Read/print formatted values

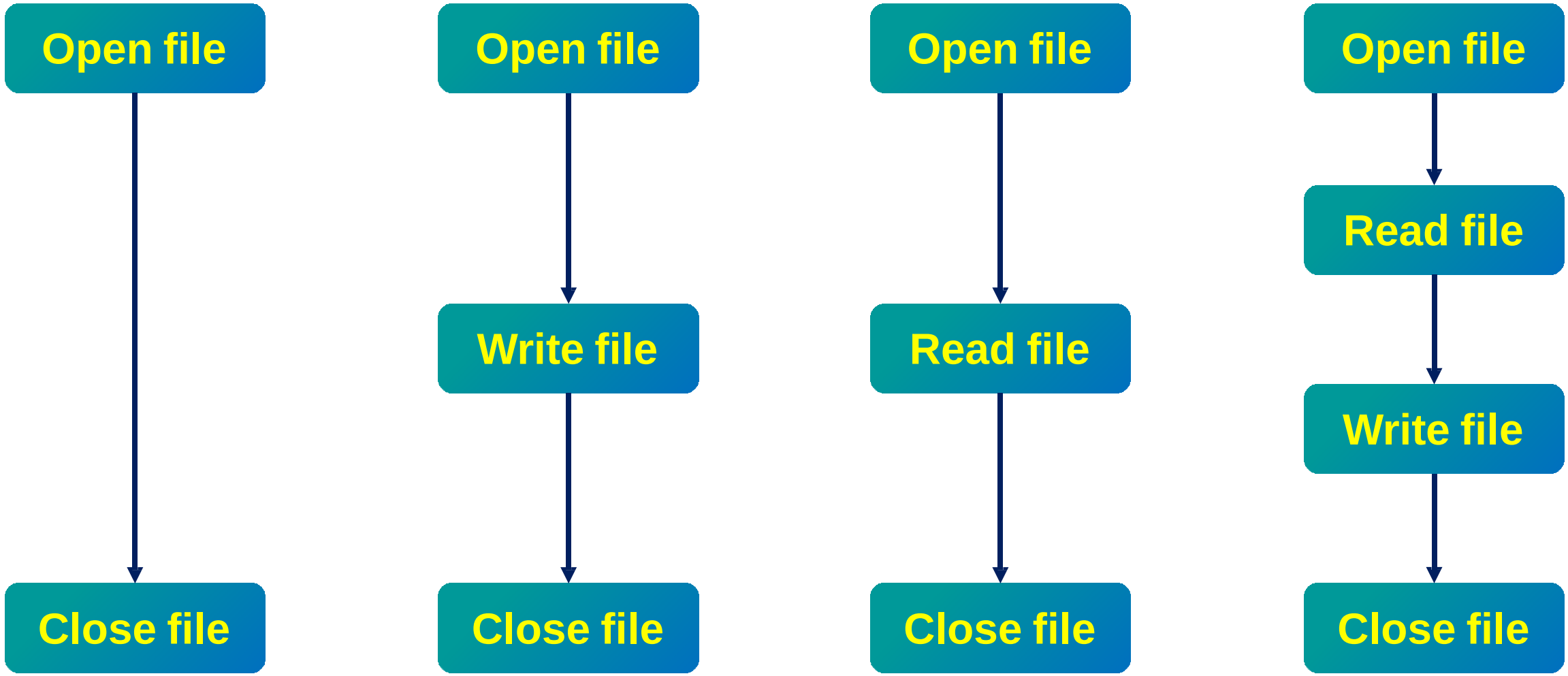
File I/O in life



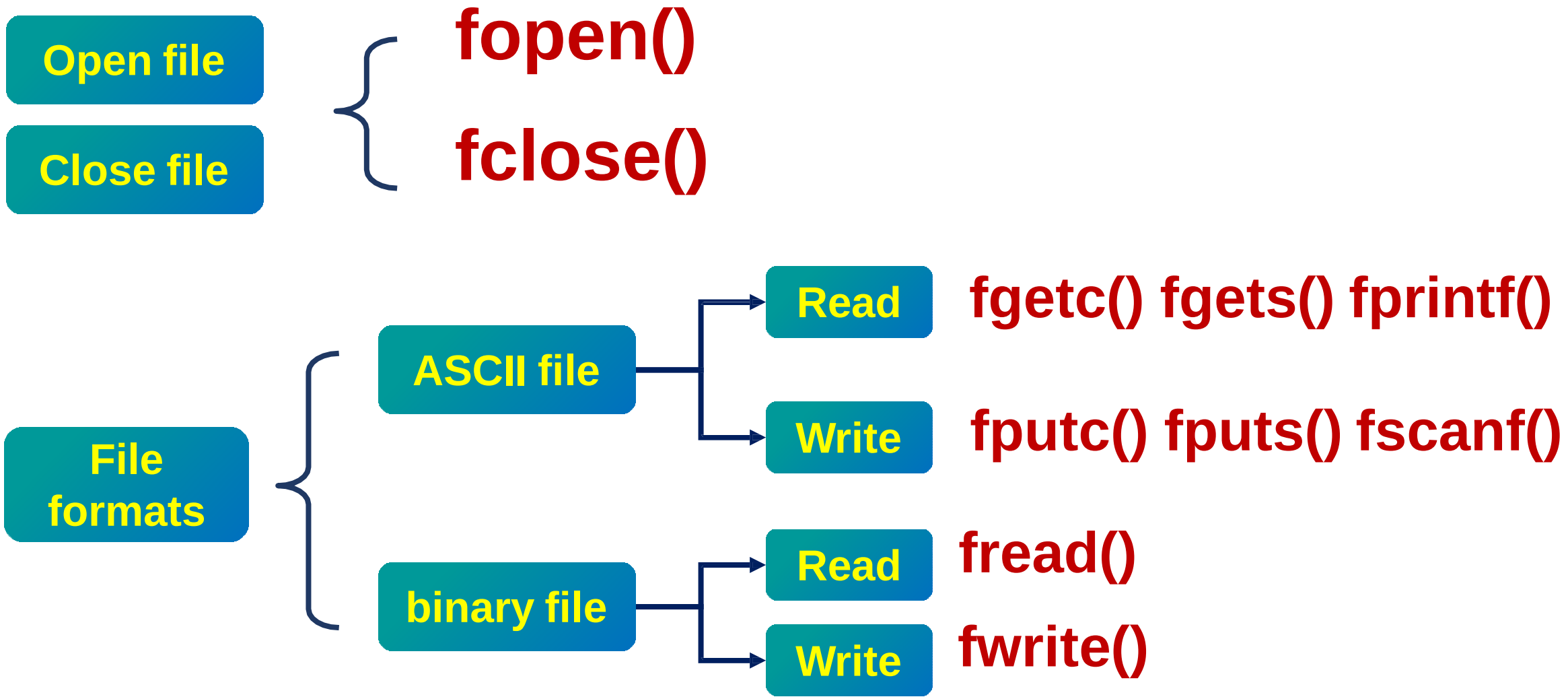
File I/O in life



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 );
```

Mode defines how you will interact with the file

Read **Write** **Append**

返回文件指针，
后续I/O操作目标。

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）

```
FILE* fp;

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


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

系统的绝对路径

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);
```

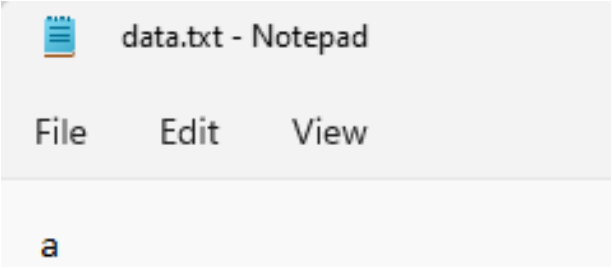
Read a single character:

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

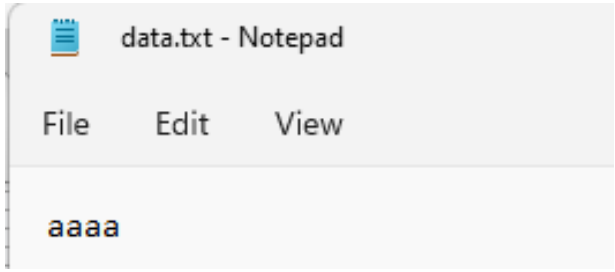
与putchar, getchar对应

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);
}
```



34

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);
}
```

Microsoft Visual Studio Debug Console

```
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
```

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>

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);
}
```

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
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:

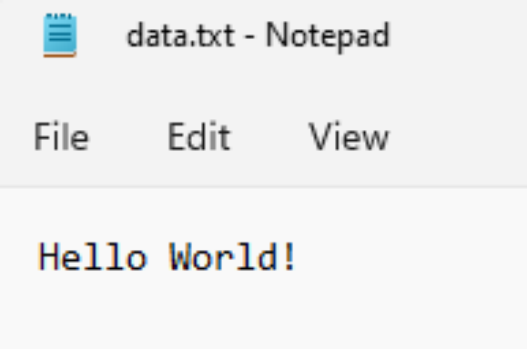
```
char* fgets(char *buf, int n, FILE *fp);
```

与puts, gets对应

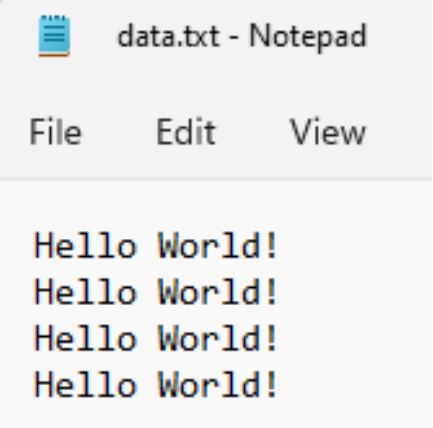
Write txt file by fputs()

```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);
}
```



- 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



Read txt file by fgets()

```c
#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, ...)
```

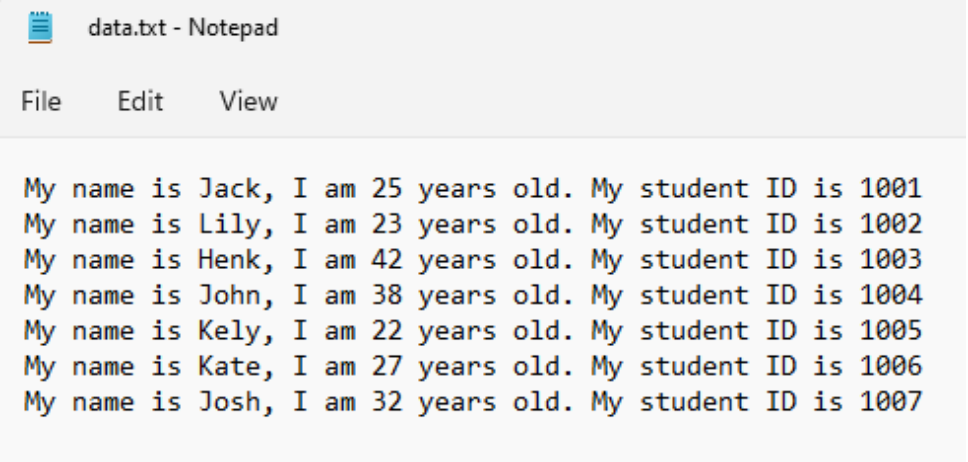
注意：
用fprintf和fscanf函数对磁盘文件读写，使用方便，容易理解，但由于在输入时要将ASCII码转换为二进制形式，在输出时又要将二进制形式转换成字符，花费时间比较多。因此，在内存与磁盘频繁交换数据的情况下，最好不用fprintf和fscanf函数，而用fread和fwrite函数。

与printf, scanf对应

Write txt file by fprintf()

```c
#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



Write txt file by fprintf()

```
#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)

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!
```

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 Studio

```
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
- getchar
- puts
- gets
- printf
- scanf
- 只能通过标准输入、标准输出、错误输出进行输入/输出

- fputc
- fgetc
- fputs
- fgets
- fprintf
- fscanf
- 通过指定设备进行输入/输出
- (stdin, stdout, stderr)

bin file

Write binary file:

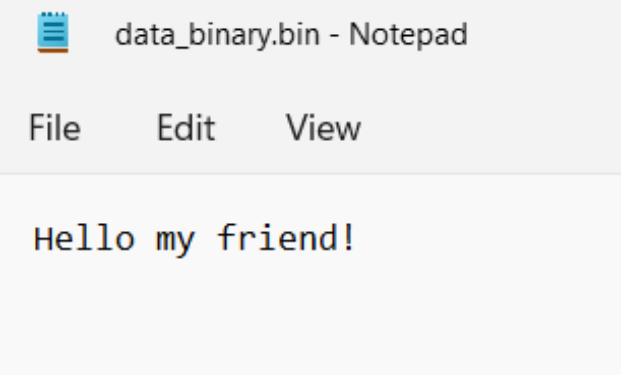
```
fwrite(const void *ptr, int size_of_elements, int
number_of_elements, FILE *fp);
```

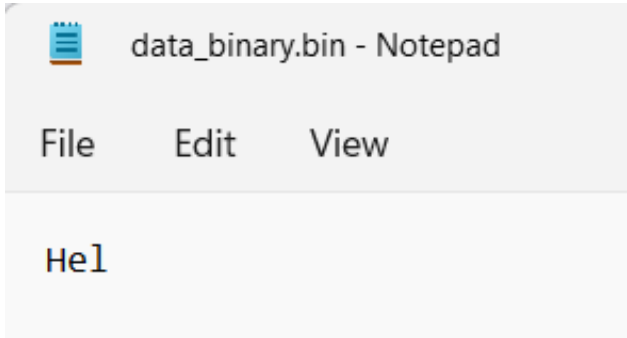
Read binary format:

```
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);
}
```





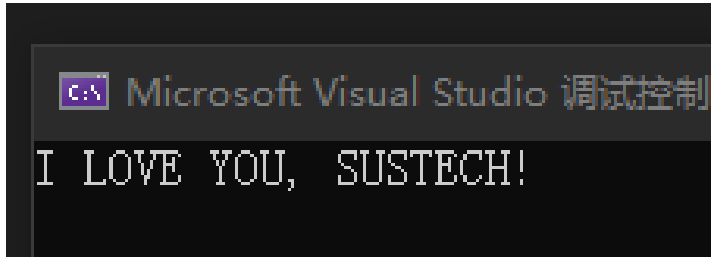
Case study

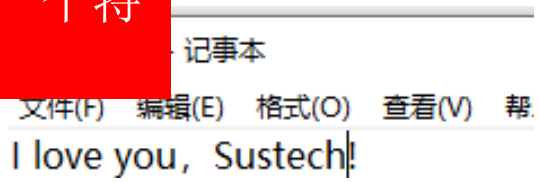
Read the file, convert lowercase to uppercase, and write to the file

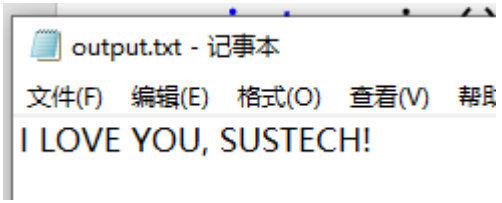
```c
int main()
{
    FILE* fp,*fp_output;
    char str[100];
    char ch;
    if ((fp = fopen("test.txt", "r")) == NULL){
        printf("file cannot open!");
        exit(0);}
    fp_output = fopen("output.txt", "w+");
    while ((ch = fgetc(fp)) != EOF) {
        if (ch >= 'a' && ch <= 'z') ch -=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

fgetc()读到文件结尾时将返回一个特殊值EOF

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







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。

Lecture 13: self-defined types

Content

1. **Struct, union, enumerate**
2. **Typedef , #define and #include**

Content

1. **Struct, union, enumerate**
2. **Typedef , #define and #include**

Structure in life

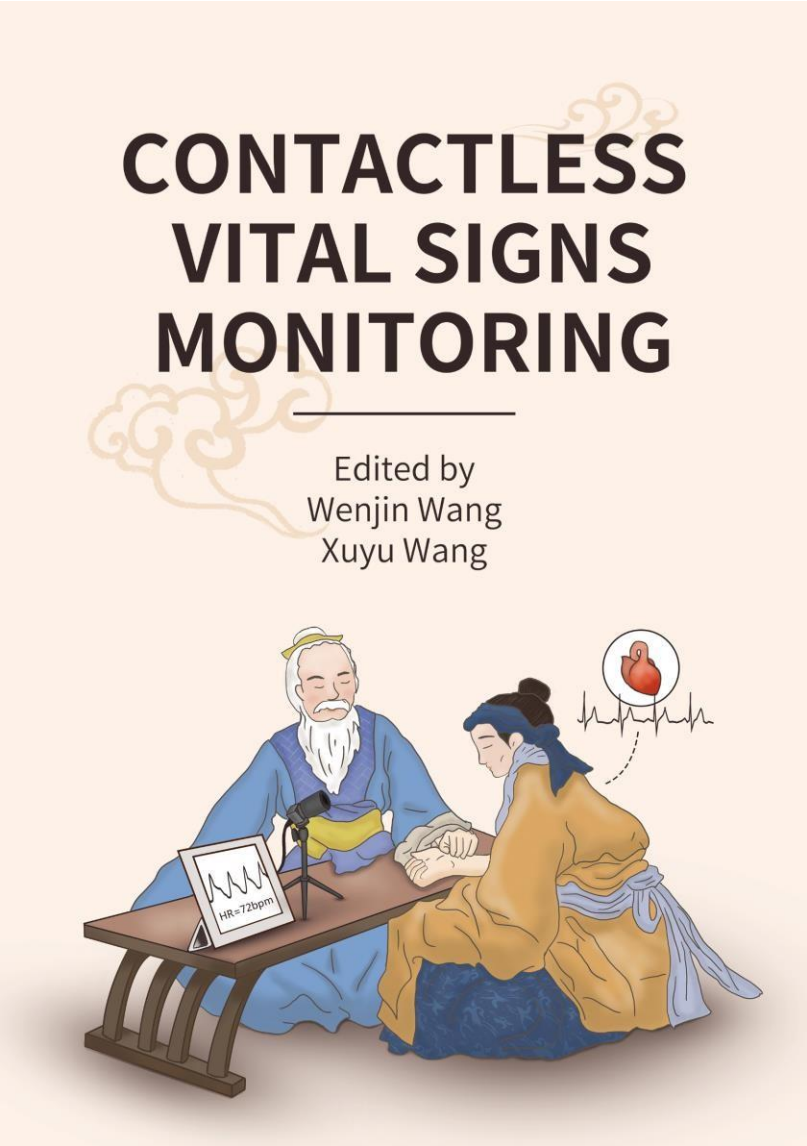
Student

- ❑ Name
- ❑ Age
- ❑ Gender
- ❑ ID
- ❑ Major
- ❑ Grade
- ❑ Birthday

Professor

- ❑ Name
- ❑ Age
- ❑ Gender
- ❑ Hat
- ❑ Paper
- ❑ Project
- ❑ Honour

Structure in life



Book

- ❑ Title
- ❑ Authors
- ❑ Publisher
- ❑ Date
- ❑ DOI
- ❑ Place
- ❑ Version

Structure in life

Patient

- ❑ Name
- ❑ Age
- ❑ Gender
- ❑ Disease
- ❑ Vital signs
- ❑ Medical records
- ❑ Symptoms

Three types of structure

Struct (结构体)	Union (共用体)	Enum (枚举型)
👍👍👍		👍
Used very often	Useless	Used but not often

Struct

You cannot use **array** to group data with different types

Struct (结构体)

Name	Age	Gender	ID	Grade	Birthday
(char[])	(int)	(char)	(int)	(float)	(char[])

What is struct?

Struct defines a new data type that allows using variables with different types.

```
struct name
{
    type        variable;
    type        variable;
    type        variable;
};
```

Struct name

Member list

How to define struct?

Student

name
age
gender
ID
grade
birthday

```
struct student
{
    char name[20];
    int age;
    char gender;
    int ID;
    int grade;
    char birthday[50];
};
```

Struct name

Member list

How to define struct?

```
#include<stdio.h>
#include<string.h>
struct student
{
    char name[20];
    int age;
    char gender;
    int ID;
    int grade;
    char birthday[50];
};

main()
{
    struct student student1;
    strcpy(student1.name, "Jack Chen");
    student1.age = 25;
    student1.gender = 'M';
    student1.ID = 123;
    student1.grade = 80;
    strcpy(student1.name, "2005-October-10");
}
```

define the struct data

(1)先声明结构体类型再定义变量名

How to define struct?

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

struct student
{
    char name[20];
    int age;  char
    gender;  int ID;
    int grade;
    char birthday[50];
} student1, student2, student3, student4;

main()
{
    strcpy(student1.name, "Jack Chen");
    student1.age = 25;
    student1.gender = 'M';
    student1.ID = 123;
    student1.grade = 80;
    strcpy(student1.birthday, "2005-October-10");
}
```

Struct variable is defined globally!

(2)在声明类型的同时定义变量

How to define struct?

(3) 直接定义结构体类型变量，可以不出现结构体名

```
#include<stdio.h>
#include<string.h>
struct
{
    char name[20];
    int age;
    char gender;
    int ID;
    int grade;
    char birthday[50];
} student1, student2, student3, student4;

main()
{
    strcpy(student1.name, "Jack Chen");
    student1.age = 25;
    student1.gender = 'M';
    student1.ID = 123;
    student1.grade = 80;
    strcpy(student1.birthday, "2005-October-10");
}
```

注意：

(1) 类型与变量是不同的概念，不要混同。只能对变量赋值、存取或运算，而不能对一个类型赋值、存取或运算。在编译时，对类型是不分配空间的，只对变量分配空间。

(2) 对结构体中的成员（即“域”），可以单独使用，它的作用与地位相当于普通变量。

(3) 成员也可以是一个结构体变量。

(4) 成员名可以与程序中的变量名相同，二者不代表同一对象。

How to define struct?

```c
#include<stdio.h>
struct student
{
    char name[20];
    int age;
    char gender;
    int ID;
    int grade;
    char birthday[50];
};
main()
{
    struct student student1 = {"Jack Chen", 25, 'M', 123, 80, "2005-October-10"};

    printf("student1 name = %s\n", student1.name);
    printf("student1 age = %d\n",  student1.age);
    printf("student1 gender = %c\n", student1.gender);
    printf("student1 ID = %d\n",     student1.ID);
    printf("student1 grade = %d\n",  student1.grade);
    printf("student1 birthday = %s\n", student1.birthday);
}
```

Initialize the struct when declaring it, must be in order!

```
student1 name = Jack Chen
student1 age = 25
student1 gender = M
student1 ID = 123
student1 grade = 80
student1 birthday = 2005-October-10
```

How to define struct?

```
#include<stdio.h>
struct student
{
    char name[20];
    int age;
    char gender;
    int ID;
    int grade;
    char birthday[50];
};
main()
{
    struct student student1 = {"Jack Chen"  , 25, 'M', 123, 80, "2005-October-10"};
    struct student student2 = {"Li Wang"    , 23, 'F', 124, 97, "2004-May-9"};
    struct student student3 = {"Steffen He" , 24, 'M', 125, 94, "2005-July-12"};
    struct student student4 = {"Tomas Huang", 25, 'M', 126, 90, "2005-March-23"};
    struct student student5 = {"Helen Luo"  , 27, 'F', 127, 84, "2005-June-15"};
}
```

Declare and define a group of students!

How to define struct?

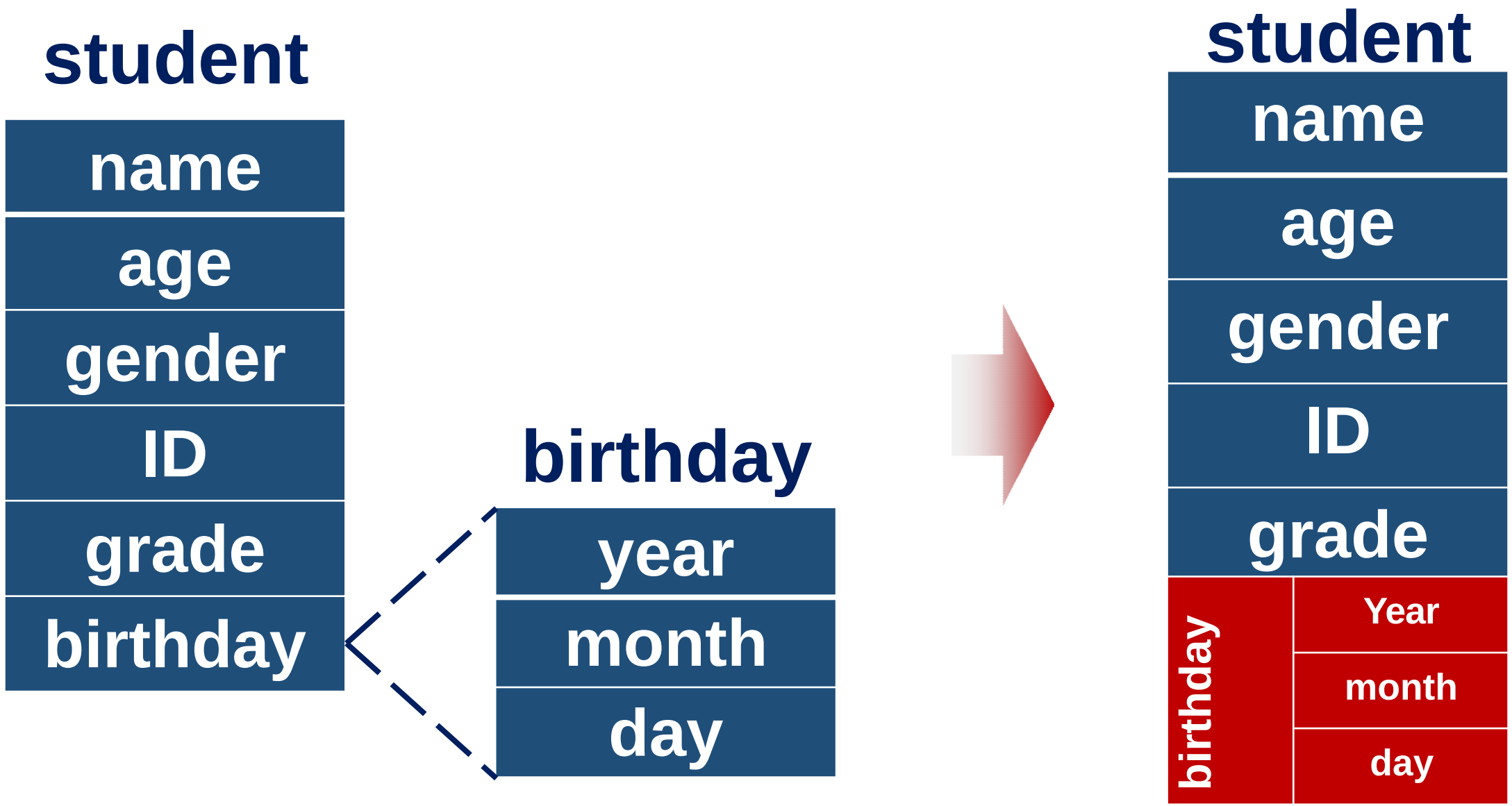
Different structs can be used in one program

```
struct student
{
    char name[20];
    int age;
    char gender;
    int ID;
    int grade;
    char birthday[50];
};
```

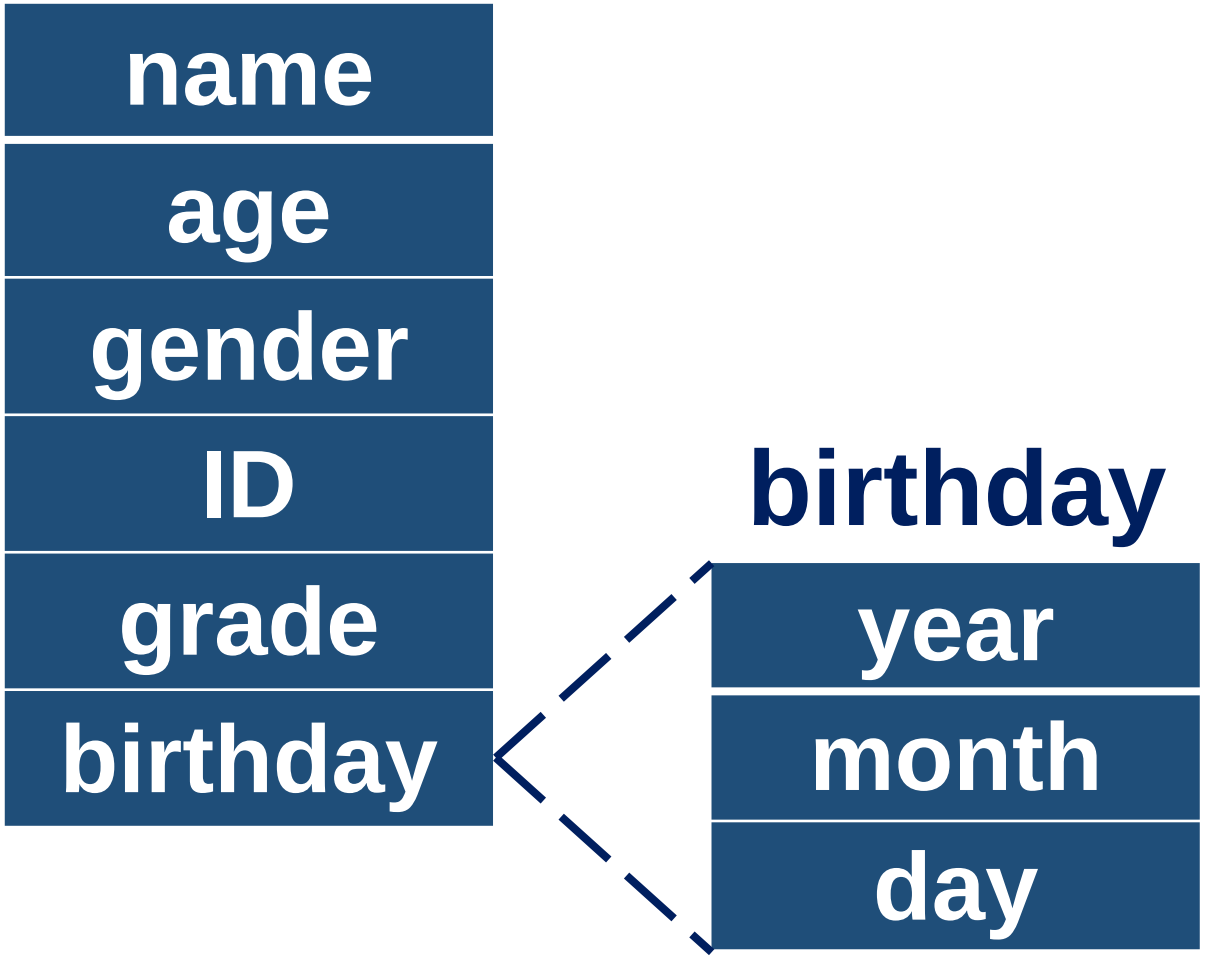
```
struct teacher
{
    char name[20];
    int age;
    char gender;
    int hat;
    int paper;
    int project ;
    int honour;
};
```

```
main()
{
    struct student student1 = {"Jack Chen", 25, 'M', 123, 80, "2005-October-10"};
    struct teacher teacher1 = {"Li Liang", 45, 'M', 1, 50, 5, 0};
}
```

Nested structs



Nested structs




```c
struct birthday
{
    int year;
    int month;
    int day;
};

struct student
{
    char name[20];
    int age;
    char gender;
    int ID;
    int grade;
    struct birthday birth;
};
```

Nested structs

```c
#include<stdio.h>

struct birthday
{
    int year;
    int month;
    int day;
};

struct student
{
    char name[20];
    int age;
    char gender;
    int ID;
    int grade;
    struct birthday birth;
};
```

```c
int main(void)
{
    struct student student1;
    strcpy(student1.name, "Jack Chen");
    student1.age = 25;
    student1.gender = 'M';
    student1.ID = 123;
    student1.grade = 80;
    student1.birth.year = 2005;
    student1.birth.month = 10;
    student1.birth.day = 10;

    printf("student1 name = %s\n", student1.name);
    printf("student1 age = %d\n",  student1.age);
    printf("student1 gender = %c\n", student1.gender);
    printf("student1 ID = %d\n",    student1.ID);
    printf("student1 grade = %d\n", student1.grade);
    printf("student1 birthday = %d-%d-
    %d\n",student1.birth.year, student1.birth.month,
    student1.birth.day);
}
```

- 如果成员本身又属一个结构体类型，则要用若干个成员运算符，一级一级地找到最低的一级的成员。
- 只能对最低级的成员进行赋值或存取以及运算。

Array of structs

① declare struct array and initialize it separately

```
struct student
{
    char name[20];
    int ID;
    char gender;
};
```

```
main()
{
    struct student stu[2];
    strcpy(stu[0].name, "Jack");
    stu[0].ID = 1;
    stu[0].gender = 'M';

   strcpy(stu[1].name, "Merry");
   stu[1].ID = 2;
   stu[1].gender = 'F';
}
```

Array of structs

② declare struct array and initialize it jointly

```
struct student
{
    char name[20];
    int ID;
    char gender;
};
```

```
main()
{
    struct student stu[2] = {
{"Jack",1,'M'}, {"Merry",2,'F'}};
}
```

Input & Output

```c
#include <stdio.h>

struct student {
        char name[20];
        int id;
        float average;
};

int main(void) {
        struct student s1[] =  { {"Kate", 1001, 90}, \
              {"Jack", 1002, 94}, {"Mike", 1003, 85} };
        FILE *p = fopen("./data.bin", "w");
        fwrite(&s1, 1, sizeof (struct student) * 3, p);
        fclose(p);
        struct student s2[3] = {0};
        FILE *p2 = fopen("./data.bin", "r");
        fread(&s2, 1, sizeof (struct student) * 3, p2);
        fclose(p2);
        printf("%s %d %f\n", s2[0].name, s2[0].id, s2[0].average);
        printf("%s %d %f\n", s2[1].name, s2[1].id, s2[1].average);
        printf("%s %d %f\n", s2[2].name, s2[2].id, s2[2].average);
        return 0;
}
```

Microsoft Visual Studio D

```
Kate 1001 90.000000
Jack 1002 94.000000
Mike 1003 85.000000
```

Pointer to structs

```
#include<stdio.h>

struct student
{
    char name[4];
    int ID;
    char gender;
};

main()
{
    struct student stu = {"Sam", 1, 'M'};

    printf("Address of stu: %x\n",    &stu);
    printf("Address of nam: %x\n",    &stu.name);
    printf("Address of ID: %x\n",     &stu.ID);
    printf("Address of gender: %x\n", &stu.gender);
}
```

You can check the address of struct!!!

stu	affafb70
name	affafb70
ID	affafb74
gender	affafb78

Pointer to structs

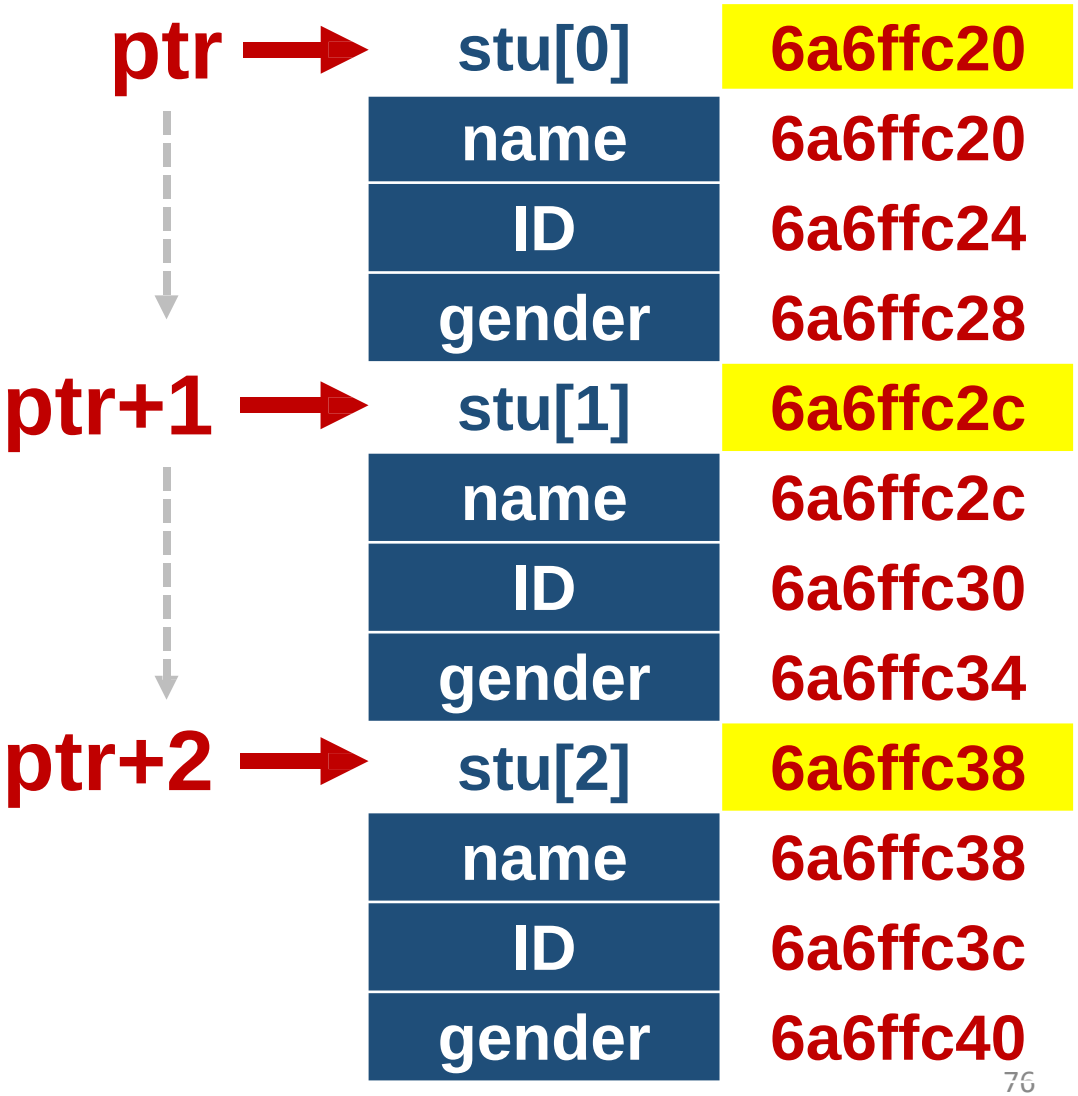
```
#include<stdio.h>

struct student
{
    char name[4];
    int ID;
    char gender;
};

main()
{
    struct student stu[2] = {{"Sam",1, 'M'},{"Jen",1, 'F'}};

    for (int i = 0; i < 2; i ++) {
        printf("Address of stu: %x\n",    &stu[i]);
        printf("Address of nam: %x\n",    &stu[i].name);
        printf("Address of ID: %x\n",     &stu[i].ID);
        printf("Address of gender: %x\n", &stu[i].gender);
    }
}
```

可以引用结构体变量成员的地址，也可以引用结构体变量的地址。

stu[0]	d3cff880
name	d3cff880
ID	d3cff884
gender	d3cff888
stu[1]	d3cff88c
name	d3cff88c
ID	d3cff890
gender	d3cff894

Pointer to structs

```c
#include<stdio.h>

struct student
{
    char name[4];
    int ID;
    char gender;
};


main()
{
    struct student stu[3] = {{"Sam",1,
    'M'},{"Jen",2, 'F'}, {"Mik", 3, 'M'}};
    struct student* ptr = stu; //&stu[0]
    printf("address of prt =   %x\n",ptr);
    printf("address of prt+1 = %x\n",ptr+1);
    printf("address of prt+2 = %x\n",ptr+2);
}
```



Pointer to structs

```
#include<stdio.h>

struct student
{
    char name[4];
    int ID;
    char gender;
};

main()
{
    struct student stu[3] = {{"Sam",1, 'M'},{"Jen",2, 'F'},
    {"Mik", 3, 'M'}};
    struct student* ptr = stu; //&stu[0];
    printf("stu 1 name = %s\n",(*ptr).name);
    printf("stu 2 name = %s\n",(*(ptr+1)).name);
    printf("stu 3 name = %s\n",(*(ptr+2)).name);

    printf("stu 1 name = %s\n",ptr ->name);
    printf("stu 2 name = %s\n",(ptr+1)->name);
    printf("stu 3 name = %s\n",(ptr+2)->name);
}
```

How to access members of struct using pointer?

① (*ptr).name
(*ptr).ID
(*ptr).gender

② ptr->name
ptr->ID
ptr->gender

Pointer to structs

以下3种形式等价：

① 结构体变量．成员名

②（＊p）．成员名

③ p->成员名

其中->称为指向运算符。

Pointer to structs

分析以下几种运算：

- p->n
- p->n++
- ++p->n

Pointer to structs

分析以下几种运算：

- p->n 得到 p 指向的结构体变量中的成员 n 的值。
- p->n++得到 p 指向的结构体变量中的成员 n 的值，用完该值后使它加 1 。
- ++p->n 得到 p 指向的结构体变量中的成员 n 的值加 1，然后再使用它。(->优先级高于++)

(++p)->n
(p++)->n

?

Struct for functions

Struct as input parameters for function

Struct as output results of function

Struct as function input

```c
#include<stdio.h>
struct student
{
    char name[5];
    int ID;
    char gender;
};
void input(struct student stu);
void main()
{
    struct student stu = {"Jack",1, 'M'};
    input(stu);
    printf("%s - %d - %c", stu.name, stu.ID, stu.gender);
}
void input(struct student stu)
{
    strcpy(stu.name, "Lily");
    stu.ID = 5;
    stu.gender = 'F';
}
```

What is value of stu?

Jack - 1 - M

or

Lily - 5 - F

- 函数间传递参数（结构struct）的值

Struct as function input

```c
#include<stdio.h>
struct student
{
    char name[5];
    int ID;
    char gender;
};
void input(struct student *stu);
void main()
{
    struct student stu = {"Jack",1, 'M'};
    input(&stu);
    printf("%s - %d - %c", stu.name, stu.ID, stu.gender);
}
void input(struct student *stu)
{
    strcpy(stu->name, "Lily");
    stu->ID = 5;
    stu->gender = 'F';
}
```

What is value of stu?

Jack - 1 - M

or

Lily - 5 - F

- 函数间传递指针（结构struct）的地址

Struct as function input

```
#include<stdio.h>
#include<string.h>
struct student
{
    char name[5];
    int ID;
    char gender;
};
void input(struct student stu[]);
int main(void)
{
    struct student stu[2];
    input(stu);
    printf("%s - %d - %c\n", stu[0].name, stu[0].ID,stu[0].gender);
    printf("%s - %d - %c",stu[1].name, stu[1].ID, stu[1].gender);

}
void input(struct student stu[])
{
    strcpy(stu[0].name, "Lily"); stu[0].ID = 5; stu[0].gender= 'F';
    strcpy(stu[1].name, "Chen"); stu[1].ID = 7; stu[1].gender = 'M';
}
```

Pass the array of structs to the function

Lily - 5 - F
Chen - 7 - M

- 函数间传递数组被当成指针

Struct as function input

```c
#include<stdio.h>
#include<string.h>
struct student
{
    char name[5];
    int ID;
    char gender;
};
struct student get();

int main(void)
{
    struct student stu = get();
    printf("%s - %d - %c", stu.name, stu.ID, stu.gender);

}
struct student get()
{
    struct student stu;
    strcpy(stu.name, "Lily"); stu.ID = 5; stu.gender = 'F';
    return stu;
}
```

Return a struct from function to main

Lily - 5 - F

Struct as function output

```
#include<stdio.h>
#include <string.h>
struct student
{
    char name[5];
    int ID;
    char gender;
};
struct student* get();
struct student stu[2];

int main(void)
{   struct student *stu=get();
    printf("%s - %d - %c", stu[0].name, stu[0].ID,stu[0].gender);
    printf("%s - %d - %c",stu[1].name, stu[1].ID, stu[1].gender);
}
struct student *get()
{
    //struct student stu[2];
    strcpy(stu[0].name, "Lily"); stu[0].ID = 5; stu[0].gender= 'F';
    strcpy(stu[1].name, "Chen"); stu[1].ID = 7; stu[1].gender = 'M';
    return stu;
}
```

Return struct array from function to main

Lily - 5 - F
Chen - 1 - M

L13_struct_pointer_arr.cpp

Case study: car model

```c
#include<stdio.h>

struct Car {
    char brand[50];
    char model[50];
    int price;
};

void main()
{
    struct Car car1 = {"Benz"  , "MAYBACH" , 5000000};
    struct Car car2 = {"Bentley" , "Flying Spur" , 4000000};
    struct Car car3 = {"Maserati", "Quattroporte" , 3000000};

    printf("%s %s %d\n", car1.brand, car1.model,  car1.price);
    printf("%s %s %d\n", car2.brand, car2.model,  car2.price);
    printf("%s %s %d\n", car3.brand, car3.model,  car3.price);
}
```

Case: if you want to buy a car after growing up?

```
Benz MAYBACH 5000000
Bentley Flying Spur 4000000
Maserati Quattroporte 3000000
```

Case study: calculate GPA

```c
#include<stdio.h>

struct course
{
char ame[20];
int score;
float credit;
};

float cal_GPA(struct course course[], int num)
{
float GPA = 0;
float weight = 0;
for (int i = 0; i < num; i++)
{
    GPA += course[i].credit * course[i].score;
    weight += course[i].credit;
}
return GPA / weight;
}
```

Case: calculate your GPA!

科目	C程序基础	高等数学	线性代数
得分	98	85	90
学分	4	5	3.5

GPA = (98 × 4 + 85 × 5 + 90 × 3.5) / (4 + 5 + 3.5) = 90.56

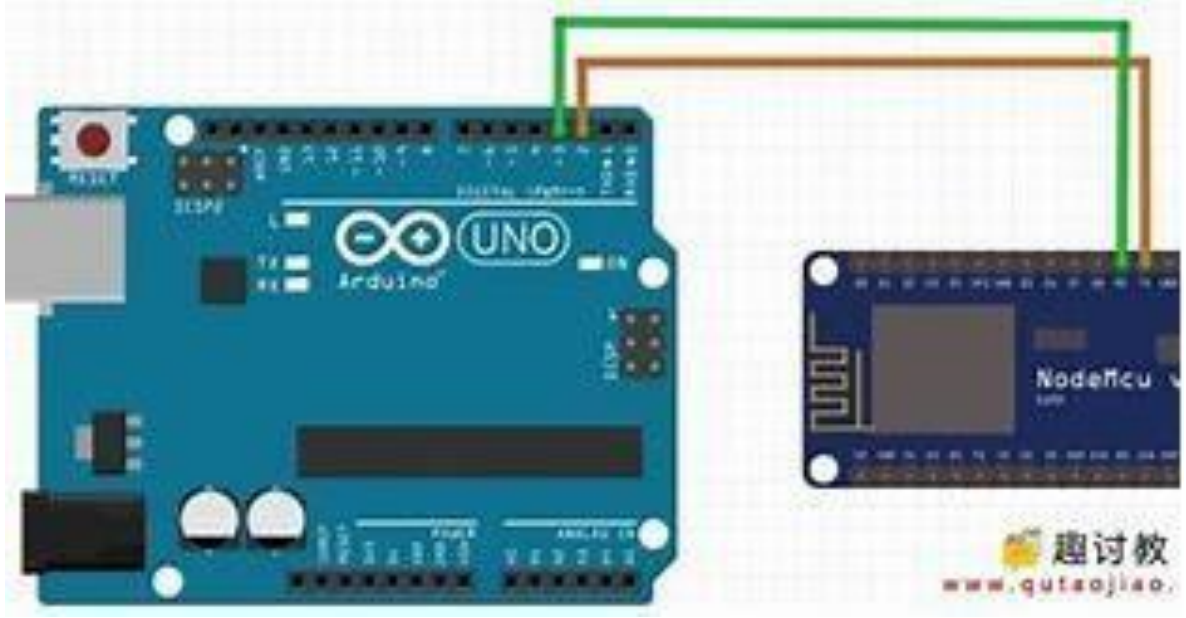
```c
main()
{
    struct course course_[3]={ { "C程序设计基础",98,4
    },{ "高等数学",85,5 }, { " 线 性 代 数 ",90,3.5 }
    };
    printf("Your GPA is %f", cal_GPA(course_, 3));
}
```

```
Your GPA is 90.559998
```

Case study: transmit packet

```c
#include<stdio.h>

struct point
{
    float x;
    float y;
    float distance;
    float verify;
};
cal_verify(struct point *in) {
    in->verify = in->x + in->y + in->distance;
}
main()
{
    struct point p = { 2.5, 2.5, 6, 0 };
    cal_verify(&p);
    printf("%f", p.verify);
}
```

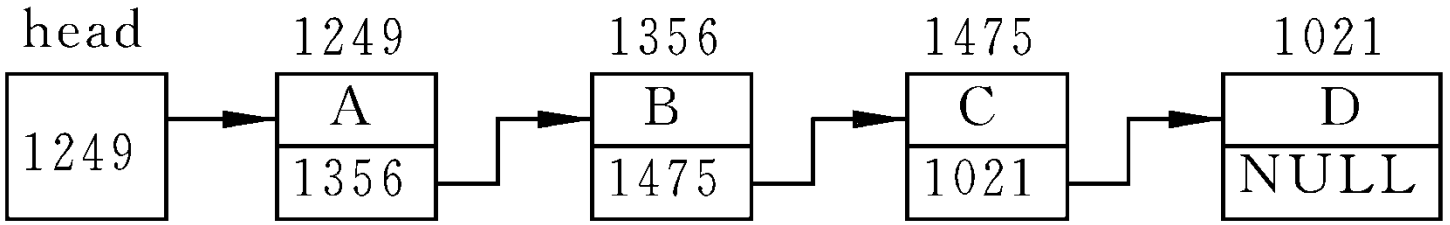
Case: verify if the transmission is correct?



Transmit 3 values (a, b, a+b), check if received values are correct

Linked List

链表是一种常见的重要的数据结构，是动态地进行存储分配的一种结构。可以根据需要开辟内存空间

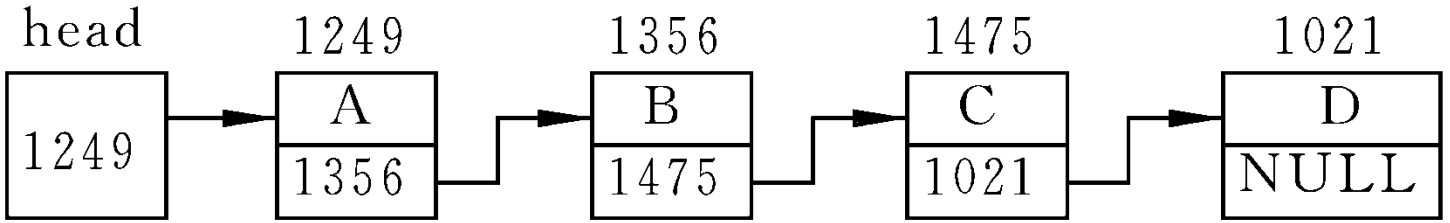


头指针：存放一个地址，该地址指向一个元素

结点：用户需要的实际数据和链接节点的指针

- 用户需要的实际数据
- 下一个节点的位置

Linked List

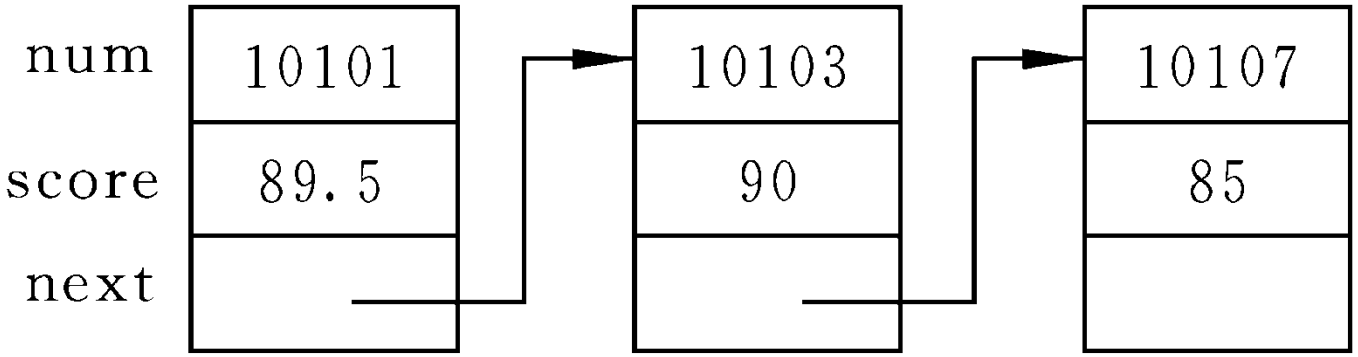


- 链表中各元素在内存中的地址可以是不连续的。要找某一元素，必须先找到上一个元素，根据它提供的下一元素地址才能找到下一个元素。**如果不提供“头指针”(head)，则整个链表都无法访问。**链表如同一条铁链一样，一环扣一环，中间是不能断开的。
- 链表这种数据结构，必须利用指针变量才能实现，即一个结点中应包含一个指针变量，用它存放下一结点的地址。

Linked List

一个结构体变量包含若干成员，这些成员可以是数值类型、字符类型、数组类型，也可以是指针类型。用指针类型成员来存放下一个结点的地址。例如：

```
struct student
    {   int num;
        float score;
        struct student *next ;};
```



Linked List

静态链表

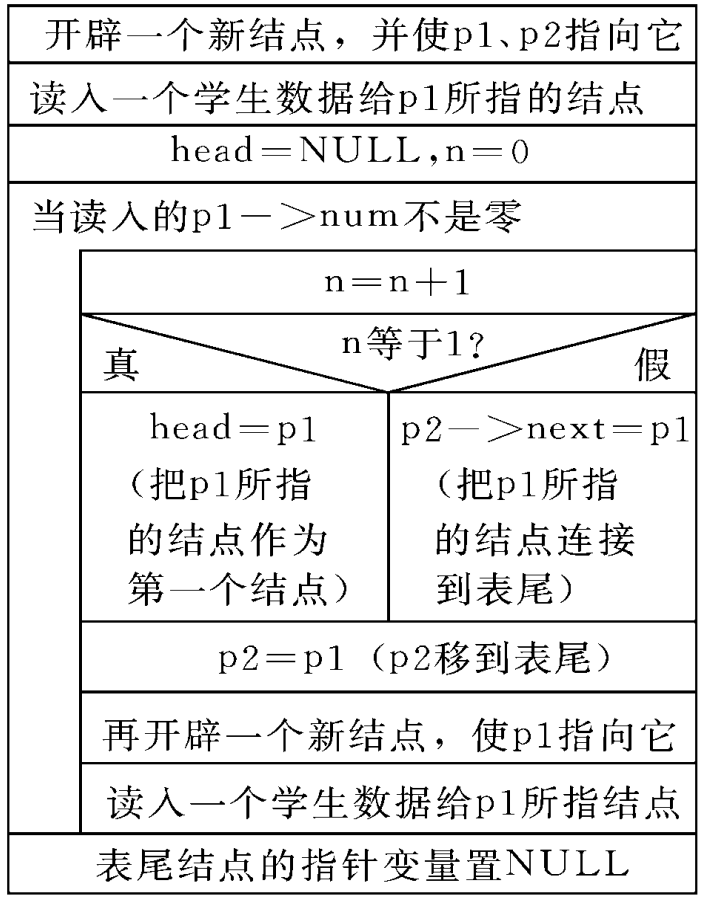
```
#include <stdio.h>
#define NULL 0
struct student {
        long num; float score; struct student *next;
};

int main(void) {
        struct student a, b, c, *head, *p;
        a. num = 99101;        a.score = 89.5;
        b. num = 99103;        b.score = 90;
        c. num = 99107;        c.score = 85;
        head = &a;       a.next = &b;
        b.next = &c;     c.next = NULL;
        p = head;
        do {
                printf("%ld %5.1f\n", p->num, p->score);
                p = p->next;
        } while (p != NULL);

}
```

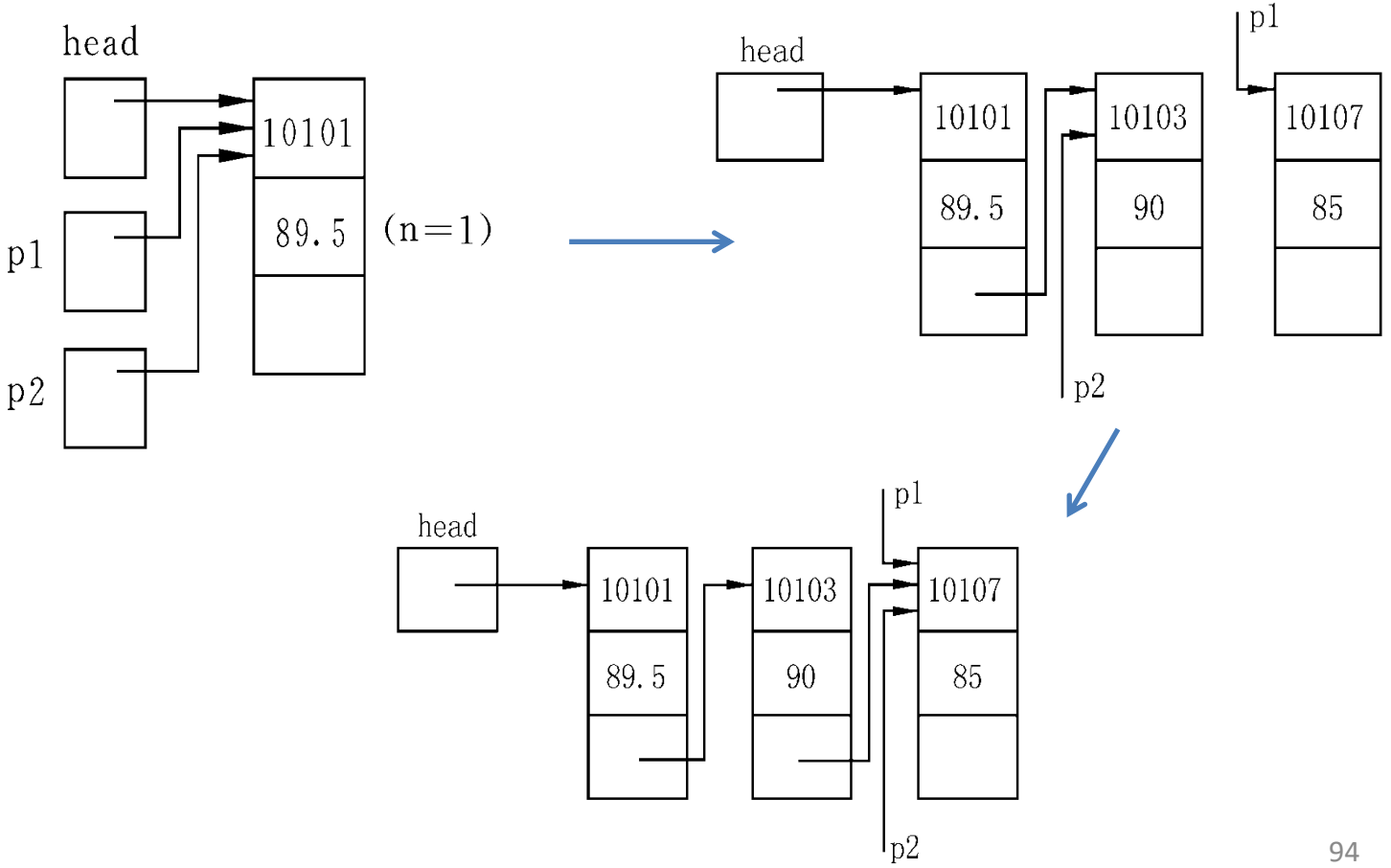
```
99101  89.5
99103  90.0
99107  85.0
```

Linked List

动态链表：所谓建立动态链表是指在程序执行过程中从无到有地建立起一个链表，即一个一个地开辟结点和输入各结点数据，并建立起前后相链的关系。

写一函数建立一个有3名学生数据的单向动态链表：





Linked List

```c
#include <stdio.h>
#include <malloc.h>
#define LEN sizeof(struct student)
struct student {
        long num;  float score;  struct student *next;
};
int n;
struct student *creat() {
        struct student *head,*p1, *p2;n = 0;
        p1 = p2 = ( struct student *) malloc(LEN);
        scanf("%ld,%f", &p1->num, &p1->score);
        head = NULL;
        while (p1->num != 0) {
                n = n + 1;
                if (n == 1)head = p1;else     p2->next = p1;
                p2 = p1;
                p1 = (struct student *)malloc(LEN);
                scanf("%ld,%f", &p1->num, &p1->score);
        }
        p2->next = NULL;return (head);}
```

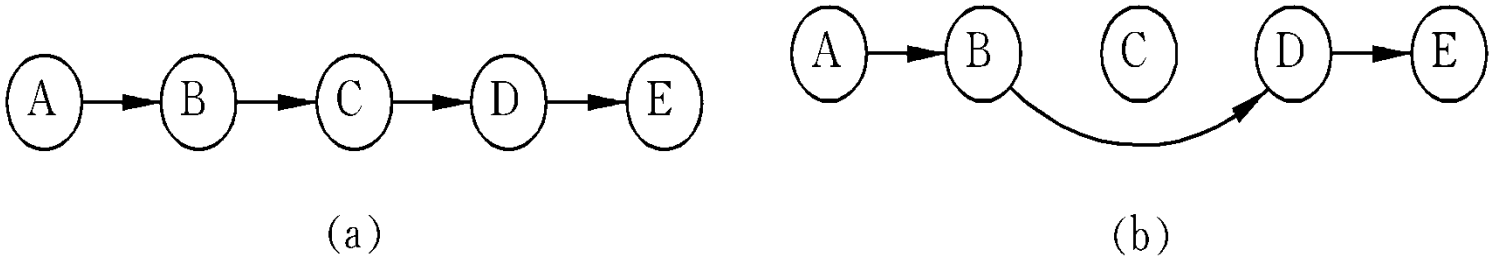
```c
int main() {
        struct student *pt;
        pt = creat();
        do {
                printf("num:%ld
score: %5.1f\n", pt->num, pt-
>score);
                pt = pt->next;
        } while (pt != NULL);

}
```

```
1001,67.5
1004,87
1003,99.5
0
num:1001    score:   67.5
num:1004    score:   87.0
num:1003    score:   99.5
```

Linked List

对链表的删除操作

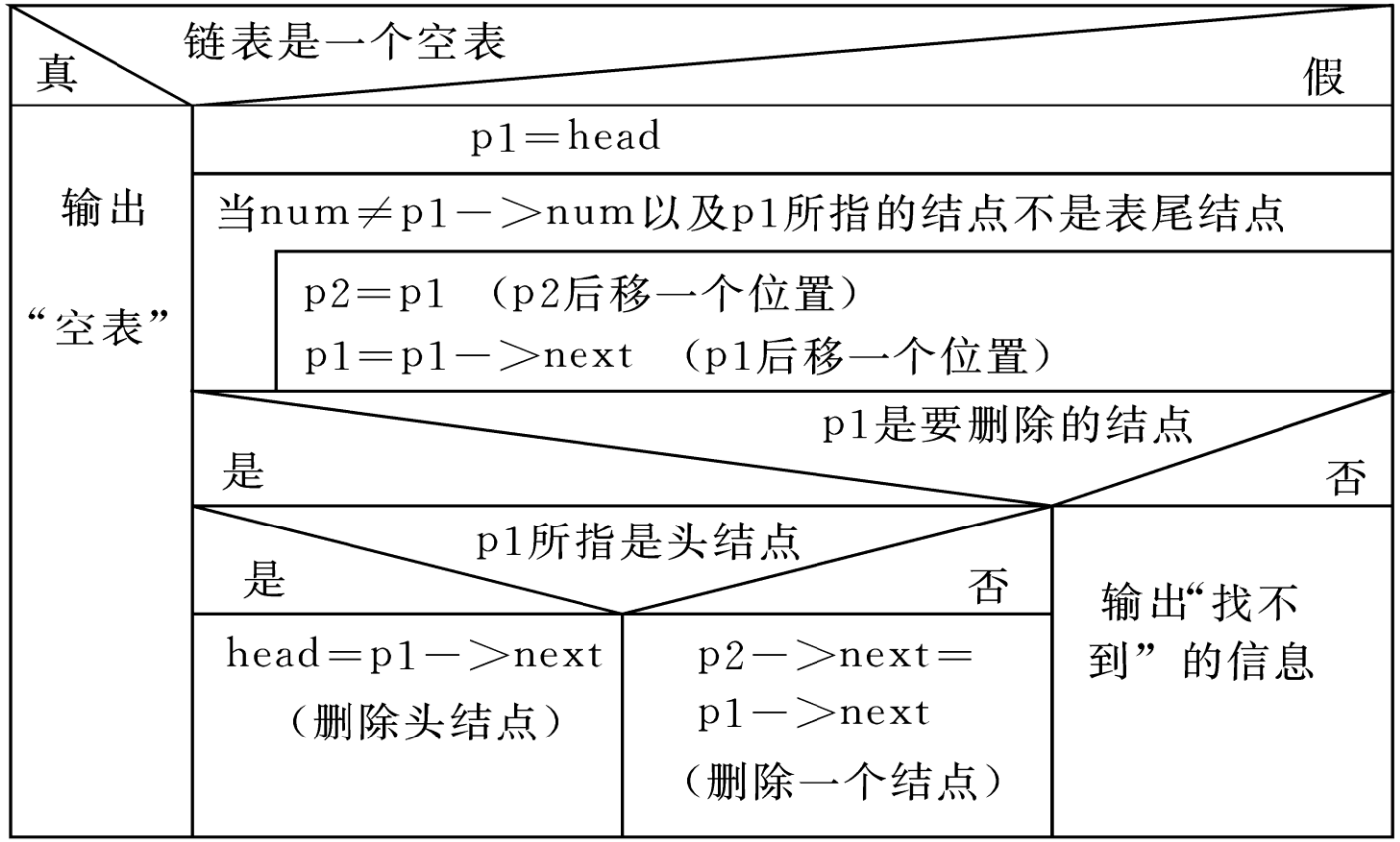
从一个动态链表中删去一个结点，并不是真正从内存中把它抹掉，而是把它从链表中分离开来，只要撤销原来的链接关系即可。



Linked List

思路:

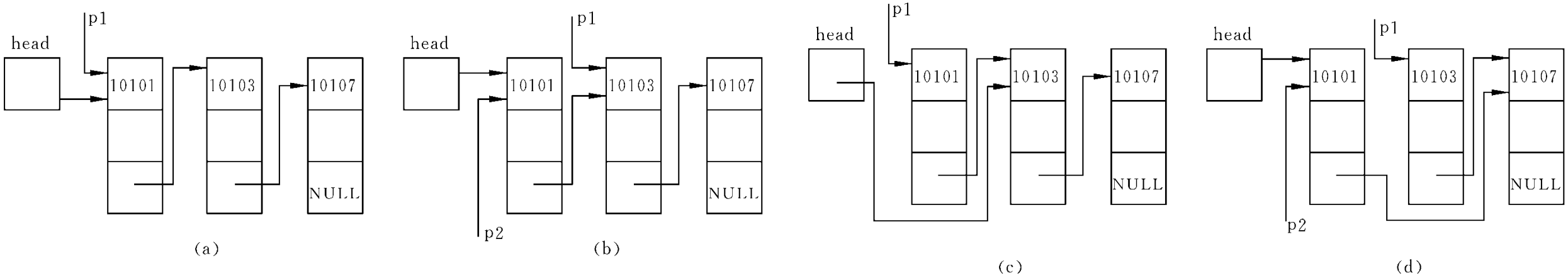
从p指向的第一个结点开始，检查该结点中的num值是否等于输入的要求删除的那个学号。如果相等就将该结点删除，如不相等，就将p后移一个结点，再如此进行下去，直到遇到表尾为止。



Linked List

1. 要删的是第一个结点（p1的值等于head的值, 图a），则应将p1->next赋给head。这时head指向原来的第二个结点。第一个结点虽然仍存在，但它已与链表脱离，因为链表中没有一个结点或头指针指向它。虽然p1还指向它，它仍指向第二个结点，但仍无济于事，现在链表的第一个结点是原来的第二个结点，原来第一个结点已“丢失”，即不再是链表中的一部分了。

2. 如果要删除的不是第一个结点，则将p1->next赋给p2->next(图d)。p2->next原来指向p1指向的结点（图中第二个结点），现在p2->next改为指向p1->next所指向的结点（图中第三个结点）。p1所指向的结点不再是链表的一部分。



Linked List

```
struct student *del(struct student *head, int num) {
        struct student *p1, *p2;
        if (head == NULL) {
                printf("\nlist null!\n");
                return (head);
        }
        p1 = head;
        while (num != p1->num && p1->next != NULL) {
                p2 = p1;
                p1 = p1->next;
        }
        if (num == p1->num) {
                if (p1 == head)
                        head = p1->next;
                else
                        p2->next = p1->next;
                printf("delete:%ld\n", num);n = n - 1;
        } else
                printf("%ld not been found!\n", num);
        return (head);
}
```

```
input records :
10101,89
10102,85
10103,98
0,0

Now,These 3 records are:
num:10101   score:  89.0
num:10102   score:  85.0
num:10103   score:  98.0
input the deleted number:10102
delete:10102

Now,These 2 records are:
num:10101   score:  89.0
num:10103   score:  98.0
```

Linked List

对链表的插入操作

对链表的插入是指将一个结点插入到一个已有的链表中。

为了能做到正确插入，必须解决两个问题：

① 怎样找到插入的位置；

② 怎样实现插入。

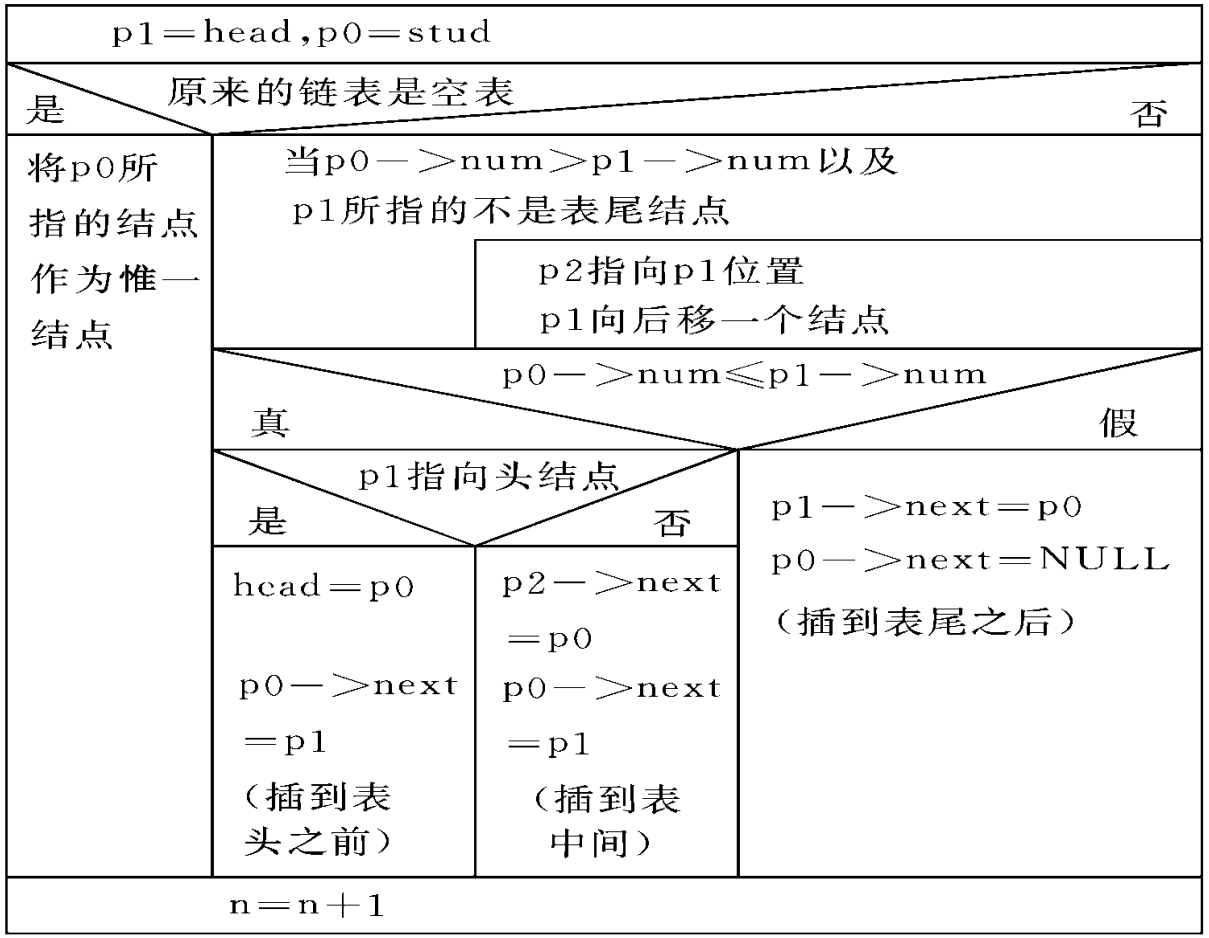
Linked List

case： 若已建立了学生链表，结点是按照其成员项（num）的值由小到大顺序排列，今要插入一个新生的结点，要求按学号的顺序插入。

思路：

先用指针变量p0指向待插入的结点，p1指向第一个结点。

将p0->num与p1->num相比较，如果p0->num>p1-> num ，则待插入的结点不应插在p1所指的结点之前。此时将p1后移，并使p2指向刚才p1所指的结点。



Linked List

```
struct student *insert(struct student *head, struct student *stud) {
        struct student *p0, *p1, *p2;
        p1 = head;          p0 = stud;
        if (head == NULL) {
                head = p0;
                p0->next = NULL;
        } else {
                while ((p0->num > p1->num) && (p1->next != NULL)) {
                        p2 = p1;p1 = p1->next;
                }
                if (p0->num <= p1->num)   {
                        if (head == p1)  head = p0;
                        else   p2->next = p0;
                        p0->next = p1;
                } else  {p1->next = p0;p0->next = NULL;} //p1->==NULL
        }
        n = n + 1;
        return (head);
}
```

```
intput records:
10101,80
10103,95
10105,92
0

Now,These 3 records are:
10101   80.0
10103   95.0
10105   92.0
intput the inserted record:
10102,90

Now,These 4 records are:
10101   80.0
10102   90.0
10103   95.0
10105   92.0
```

When to use struct?

When you want to group different types of data in a single unit!

Union

Union defines a new data type that allows using variables with different types at the **<u>same memory location</u>**!

```
union [union tag]
{
    type variable;
    type variable;
    …
};
```

Union (共用体)

Union

```
struct student
{
   char name[20];
   int ID;
   int grade;
};
```

```
union student
{
   char name[20];
   int ID;
   int grade;
};
```

sizeof(student) = 20

20 bytes = max(20, 4, 4)

Store at the same memory location!!!

Union versus struct

```
#include <stdio.h>

union Data
{
    int i;
    float f;
    char str[20];
};

main( )
{
    union Data data;
    data.i = 10;
    data.f = 220.5;
    strcpy(data.str, "C Programming");
    printf( "data.i : %d\n", data.i);
    printf( "data.f : %f\n", data.f);
    printf( "data.str : %s\n", data.str);
}
```

```
data.i : 1917853763
data.f :
4122360580327794860452
759994368.000000
data.str : C
Programming
```

```
#include <stdio.h>

struct Data
{
    int i;
    float f;
    char str[20];
};

main( )
{
    struct Data data;
    data.i = 10;
    data.f = 220.5;
    strcpy(data.str, "C Programming");
    printf( "data.i : %d\n", data.i);
    printf( "data.f : %f\n", data.f);
    printf( "data.str : %s\n", data.str);
}
```

```
data.i : 10
data.f : 220.500000
data.str : C
Programming
```

Union versus struct

```
#include <stdio.h>

union Data
{
    int i;
    float f;
    char str[20];
};

main( )
{
    union Data data;
    data.i = 10;
    printf( "data.i : %d\n", data.i);
    data.f = 220.5;
    printf( "data.f : %f\n", data.f);
    strcpy(data.str, "C Programming");
    printf( "data.str : %s\n", data.str);
}
```

```
data.i: 10
data.f: 220.500000
data.str: C Programming
```

Union

- 同一个内存段可以用来存放几种不同类型的成员，但在每一瞬时只能存放其中一种，而不是同时存放几种。

```
union Date {
        int i;
        char ch;
        float f;
} a;
```

```
void main() {
        a.i = 97;
        printf("%d\n", a.i);
        printf("%c\n", a.ch);
        printf("%f", a.f);
}
```

```
97
a
0.000000
```

- 共用体变量中起作用的成员是最后一次存放的成员，在存入一个新的成员后原有的成员就失去作用。
- 共用体变量的地址和它的各成员的地址都是同一地址。
- 不能把共用体变量作为函数参数，也不能使函数带回共用体变量，但可以使用指向共用体变量的指针。C99允许用共用体变量作为函数参数

Union

- 不能对共用体变量名赋值，也不能企图引用变量名来得到一个值，又不能在定义共用体变量时对它初始化。

```
a = 1;
m = a;
b = a;
```

×不能对共用体变量赋值，赋给谁？

× 企图引用共用体变量名以得到一个值赋给整型变量m

√ a和b是同类型的共用体变量，合法

- 共用体类型可以出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可以出现在共用体类型定义中，数组也可以作为共用体的成员。

case

设有若干个人员的数据，其中有学生和教师。学生的数据中包括：姓名、号码、性别、职业、班级。教师的数据包括：姓名、号码、性别、职业、职务。可以看出，学生和教师所包含的数据是不同的。现要求把它们放在同一表格中。

num	name	sex	job	class(班) / position(职务)
101	Li	f	s	501
102	Wang	m	t	prof

```
struct {
        int num;
        char name[10];
        char sex;
        char job;
        union {
                int banji;
                char position[10];
        } category;
} person[2];
```

case

```
void main() {
	int i;
	for (i = 0; i < 2; i++) {
		scanf("%d %s %c %c", &person[i].num, &person[i].name, &person[i].sex,
&person[i].job);
		if (person[i].job == 's')		scanf("%d", &person[i].category.banji);
		else if (person[i].job == 't')	scanf("%s", person[i].category.position);
		else		printf("Input error!");
		}
	printf("\n");
	printf("No. name sex job class/position\n");
	for (i = 0; i < 2; i++) {
		if (person[i].job == 's')
			printf(" % 6d % 10s % 3c % 3c % 6d\n", person[i].num,
				person[i].name, person[i].sex, person[i].job,person[i].category.banji);
		else
			printf(" % 6d % 10s % 3c % 3c % 6s\n", person[i].num,person[i].name,
		person[i].sex,  person[i].job,person[i].category.position);
		}}
```

```
101 Li f s 501
102 Wang m t prof

No.  name sex job class/position
    101        Li   f   s    501
    102      Wang   m   t   prof
```

When to use union?

Do NOT use union, use struct as much as you can!!!

Enumerate

Enum is a user defined data type in C, **<u>assign names to integer constants</u>**, for a program easy to read and maintain.

```
enum [union tag]
{
      variable;
      variable;
     …
};
```

All integers by default!

Enum (枚举型)

Enumerate

```
enum week { Mon, Tue, Wed, Thu, Fri, Sat, Sun}
```

0 1 2 3 4 5 6

默认从0开始

```
enum week { Mon=1, Tue, Wed, Thu, Fri, Sat,
Sun}
```

1 2 3 4 5 6 7

```
enum week day;
day = Mon;
```

Enumerate

Enum assigns names to integer constants

```c
#include<stdio.h>

enum week { Mon, Tue, Wed, Thu, Fri, Sat,
Sun};

main()
{
    enum week day;

    for (day = Mon; day <=Sun; day++)
    {
        printf("%d\n", day);
    }
}
```

```c
#include<stdio.h>

enum week { Mon, Tue, Wed, Thu, Fri,
Sat, Sun};

main()
{
    for (int i = Mon; i <=Sun; i++)
    {
        printf("%d\n", i);
    }
}
```

.c支持
.cpp不支持

Enumerate

Enum assigns names to integer constants

```c
#include<stdio.h>

enum Year { Jan, Feb, Mar, Apr, May,June,
July, Aug, Sept, Oct, Nov, Dec};

main()
{
    enum Year year;

    for (year = Jan; year <= Dec; year++)
    {
        printf("%d\n", year);
    }
}
```

```c
#include<stdio.h>

enum Year { Jan, Feb, Mar, Apr, May,June,
July, Aug, Sept, Oct, Nov, Dec};

main()
{

    for (int i = Jan; i <= Dec; i++)
    {
        printf("%d\n", i);
    }
}
```

Case study: check weekday

```c
#include <stdio.h>

main() {
    enum week { Mon = 1, Tues, Wed,Thurs,
    Fri, Sat, Sun } day;
    scanf_s("%d", &day);
    switch (day) {
    case Mon: puts("Monday"); break;
    case Tues: puts("Tuesday"); break;
    case Wed: puts("Wednesday"); break;
    case Thurs: puts("Thursday"); break;
    case Fri: puts("Friday"); break;
    case Sat: puts("Saturday"); break;
    case Sun: puts("Sunday"); break;
    default: puts("Error!");
    }
}
```

Case: input a number, check the weekday

```
4
Thursday
```

```
6
Saturday
```

```
15
Error!
```

Case study: check season

```c
#include <stdio.h>

enum Season{spring, summer, fall,  winter};

main()
{
    enum Season now = spring;

    if (now < summer)
    {
        printf("It's spring now");
    }
}
```

Case: is it spring now?

```
It's spring now
```

When to use enum?

When you want to assign a sequential of names with integers

- 👍 `Jan, Feb, Mar, Apr, May, June, July, Aug, Sept, Oct, Nov, Dec`
- 👍 `Mon, Tue, Wed, Thu, Fri, Sat, Sun`
- 👎 `Jack, Lily, Tom, John, Wim, Kevin, Henk`

Summary of three data types

Struct (结构体)	Union (共用体)	Enum (枚举型)
👍👍👍		👍
Used very often	Very useless	Used but not often