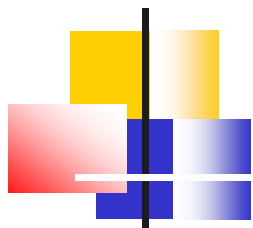


# Welcome

## C库函数

### 动态内存分配、文件操作

北京亚嵌教育研究中心  
©2011 AKAE

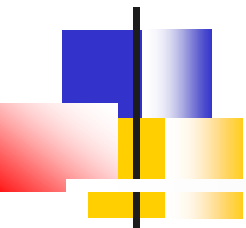


# 本次课程内容大纲

---

- 动态内存分配
- 文件操作
- 带缓冲的文件及其类别

## Section 1



# C标准库

## 内存分配库函数

# 内存分配函数

## ■ 堆空间分配函数

### ■ void \*malloc(size\_t size)

■ 功能：动态申请内存

■ 参数解释：**size**代表要申请的内存字节数

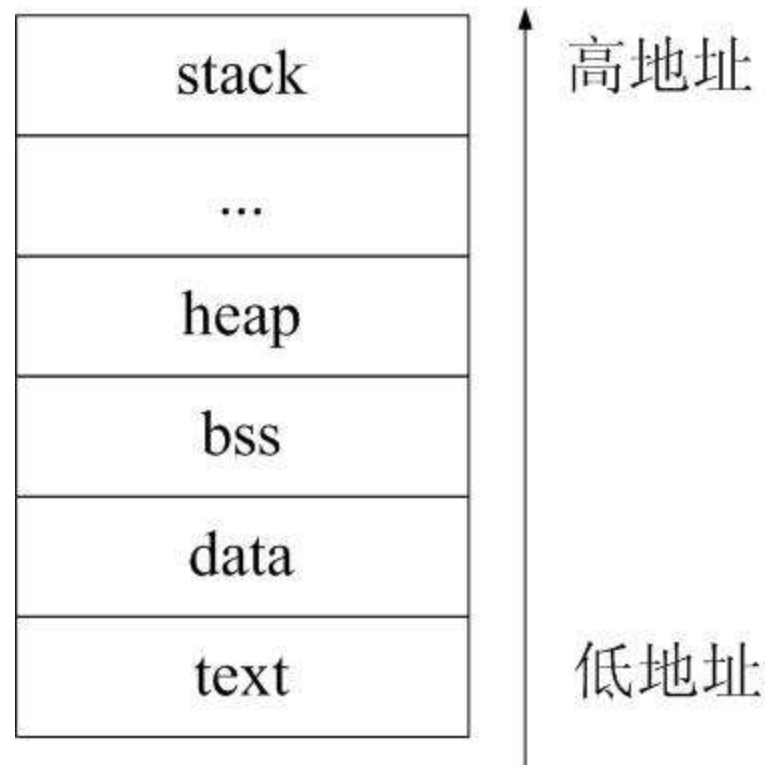
■ 返回值：**成功**：返回从堆区申请到的**size**个字节的内存的首地址

**失败**：返回NULL，申请空间失败

## 内存分配函数

■ 各个短的内存布局

- **stack**: 栈区可用的内存相对较小
- **heap**: 堆区可用空间较大



# 内存分配函数

## ■ 堆空间释放函数

## ■ void free(void \*ptr);

■ 参数解释: ptr只能是动态申请到的内存块的首地址

■ 功能: 释放动态申请的内存

■ free应该和malloc等函数成对出现

# 内存分配函数

## ■ 堆空间释放函数

■ `void free(void *ptr);`

■ `free`函数出错原因:

■ 释放的不是动态申请的内存

■ `ptr`不代表动态内存的首地址

# 内存分配函数

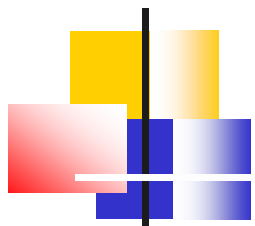
## ■ 堆空间释放函数

### ■ `void free(void *ptr);`

- 保证了资源的及时清理

- 释放内存后，原来申请的内存空间不能再用，`ptr`被称作野指针，不可再有`*ptr = ...`的操作，通常在释放后会将`ptr = NULL`





## 内存分配函数

### ■ 堆空间申请、释放函数使用基本流程

■ Step1: `p = malloc(size);`

■ Step2: 使用申请到的内存

■ Step3: `free(p);`

p的值在第2步中没有修改

■ Step4: `p = NULL;`

# 内存分配函数

## ■ 堆空间分配函数

## ■ `void *calloc(size_t nmemb, size_t size)`

■ 参数：nmemb: 申请存放nmemb个元素的空间

size: 每个元素占用size字节

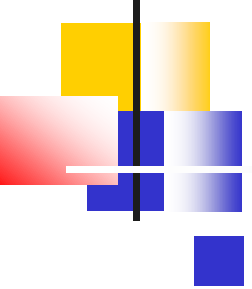
申请的空间为：size \* nmemb字节

■ 返回值：与malloc返回值相同

# 内存分配函数

- 堆空间分配函数
- `void *calloc(size_t nmemb, size_t size)`
  - 将申请`nmemb*size`字节
  - 将申请的每个字节清零

# 内存分配函数



练习：

使用malloc函数实现calloc

# 内存分配函数

## ■ 堆空间分配函数

## ■ `void *realloc(void *ptr, size_t size)`

■ 功能：修改占用的动态内存大小

■ 参数：**ptr**:只能是曾经动态获取的内存块的首地址

**size**: 修改后的动态内存字节数

■ 返回值：与**malloc**相同

# 内存分配函数

## ■ 堆空间分配函数

## ■ `void *realloc(void *ptr, size_t size)`

- 该函数不仅能修改内存大小，还可以将原有内存中的数据复制到新的内存块中来
- 重新获取的内存地址上连续
- 缺点：`realloc`效率较低

## 内存分配函数

- 堆空间分配函数
- `void *realloc(void *ptr, size_t size)`
- 两个特殊用法:
  - `realloc(NULL, size) === malloc(size)`
  - `realloc(ptr, 0) === free(ptr)`

# 内存分配函数

- 课程练习
- 输入若干自然数，若输入数据为0则结束输入，把以上数据保存到堆中，然后打印。



# 内存分配函数

## 课程练习

- 输入一个句子，统计句子中各个单词出现的次数，如：句子“hello world hello hello”，输出“hello”出现3次，“world”出现1次

```
typedef struct{  
    char *key;  
    int freq;  
}word_t;
```

- 要求成员key指向的内存存储单词的内容，成员freq记录该单词出现的次数

## Section 2

# 文件

---

# 文件

■ **文件：** 二进制数据，存储在磁盘上

■ **文本文件：** 文件的每个字节都是可显示的ASCII，主要为了给人阅读，常见的文本文件：.c源文件，文档，网页

■ **二进制文件：** 存储原始的二进制数据，不能解释为ASCII，二进制文件用于存储数据，计算机容易识别，常见的二进制文件：电影、图片、目标文件、可执行文件

# 文件

- 文件存储在磁盘上，磁盘的访问速度远远低于内存
  - 对磁盘的访问时间是磁盘振臂的机械操作时间以及数据传输时间的累加和
  - 磁盘访问时间：3-15ms
  - 内存访问时间：100-150ns

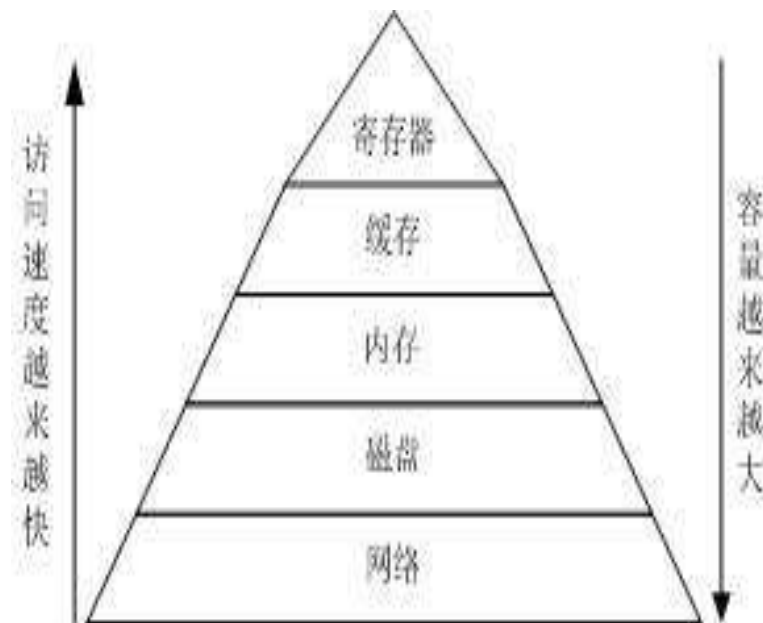
# 文件

## 计算机的分层存储结构

### ■ 上一层是下一层的缓冲

■ 如：把磁盘文件缓冲到内存中来

■ 减少频繁访问慢速设备：预读入、延迟写



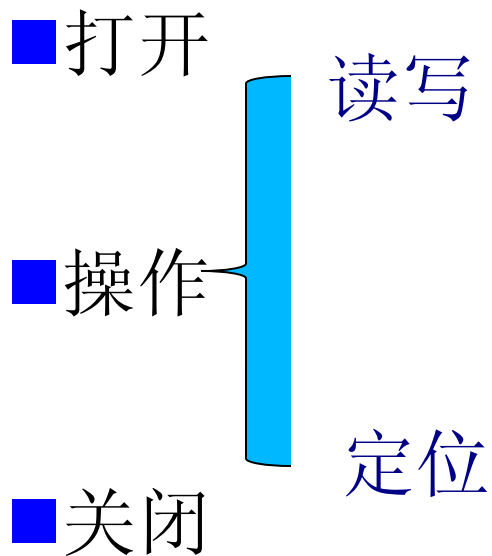
## Section 3

# 文件操作

## 打开、关闭

# 文件操作

■ 文件操作：通过库函数实现对文件的操作



# 文件打开

## 文件操作：

### ■ 打开

`FILE *fopen(const char *path, const char *mode)`

■ **FILE**：文件操作结构体，包含文件在内核中的标识以及文件读写位置等信息

■ 不需要操作FILE结构体中的成员



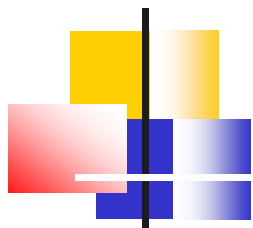
# 文件打开

`FILE *fopen(const char *path, const char *mode)`

■ `path`: 文件路径

■ `mode`: 文件操作模式

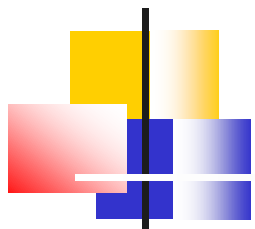
模式	含义
“r”	只读，文件必须存在
“w”	只写，如果文件不存在则创建，若文件存在则清空重写
“a”	在文件尾追加，文件不存在则创建
“r+”	读写，文件必须存在
“w+”	读写，如果文件不存在则创建，若文件存在则清空重写
“a+”	读和追加，如果文件不存在则创建



# 文件打开

`FILE *fopen(const char *path, const char *mode)`

- 打开文件成功则返回文件指针
  - 输入输出设备也被作为文件处理，相应的文件指针为：stdin、stdout、stderr，可直接使用
- 打开文件失败则返回NULL并设置errno



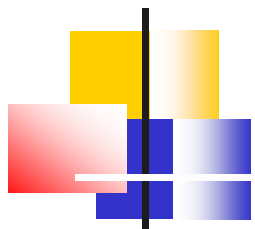
# 文件打开

`FILE *fopen(const char *path, const char *mode)`

■ `errno`是一个全局变量，用于记录最近一次系统函数的出错原因，如果系统调用出错则`errno`中保存一个整数表示错误号，使用`errno`时应包含头文件`errno.h`

■ `void perror(const char *s) //stdio.h`

■ `char *strerror(int errnum) //string.h`



# 文件打开

- `char *strerror(int errnum) //string.h`
  - 功能：把错误号转化为容易识别的文字描述
  - 参数：errnum表示错误号
  - 返回值：表示文字描述的首地址

# 文件打开

■ `void perror(const char *s) //stdio.h`

■ 功能：把错误号转化为容易识别的文字描述

■ 参数：s是调用者需要显示的内容

■ `perror`打印错误描述打印结构为：

s: 错误的文字描述

s是传入的字符串的内容

冒号和文字描述是`peror`添加的内容

`perror`把错误原因打印到标准错误输出文件中

# 文件关闭

## ■ 文件操作:

### ■ 关闭

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

### ■ 关闭fp标识的文件

■ fclose应该和fopen成对出现，文件需要关闭，主要的作用是回收系统资源并完成将缓冲区数据会写到磁盘

### ■ 关闭本地文件是一般不判断fclose的返回值

# 文件关闭

## ■ 隐式回收系统资源

- 进程运行正常结束，由系统自动回收进程占用的资源，故进程打开的文件会被隐式回收
- malloc分配的动态内存也可被隐式回收
- 隐式回收的前提是：进程正常结束
- 编程中最好主动完成资源释放，不要依赖于系统的隐式回收

## Section 3



# 文件操作 读写文件

---



# 文件读写

## ■ 文件操作：

### ■ 读写函数注意事项

- 读操作函数的要求文件的打开方式是可读
- 文件打开时，读写位置在文件起始处，每调用一次读写函数读写位置会相应地移动（读写函数完成）

# 文件读写

## ■ 文件操作:

### ■ 读写

#### 1. 按字节读写文件

`int fgetc(FILE *fp)    int getchar(void)`

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

成功返回读到的字节，出错或到达文件尾返回EOF

fgetc返回值为int型，为了能够接收文件结束标记  
EOF（#define EOF (-1)）

# 文件读写

## ■ 文件操作：

文件结束标记：EOF（#define EOF -1）

EOF不是文件的内容

fgetc函数的返回值应使用为int型变量接收而不是unsigned char型，因为如果fgetc读到文件尾将返回EOF，EOF是int型的-1。

# 文件读写

## ■ 文件操作:

### ■ 读写

#### ● 按行读写文件

`char *fgets(char *buf, int size, FILE *fp)`

成功返回时返回值与buf的值相同，出错或读到文件尾返回为NULL

# 文件读写

## ■ 文件操作:

### ■ 读写

#### ● 按行读写文件

`char *fgets(char *buf, int size, FILE *fp)`

函数返回条件:

- 1、读到size-1个有效字符
- 2、读取到'\n'字符，并将'\n'字符存入buf

`Char *gets(char *s):`尽量少用

# 文件读写

## 文件操作：

### ■ 读写

#### ● 按行读写文件

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

将字符串中'\0'前的有效内容写入文件并不会因s指向的字符串中存在'\n'而只向文件存入'\n'之前的内容："abc\nab"，存入文件的就是abc\nab各个字符，对于fputs来说'\n'不是特殊字符

# 文件读写

## ■ 文件操作：

### ■ 读写

#### ● 按行读写文件

`int puts(const char *s)`

将字符串中'\0'之前的有效内容写到标准输出，随后自动写一个'\n'到标准输出。

# 文件读写

## ■ 文件操作：

### ■ 读写

- 按行读写文件的函数一般用于操作文本文件

■ 练习：从键盘上输入字符串然后将小写字母转换为大写字母后写入文件



# 文件读写

## ■ 文件操作:

### ■ 读写

#### ● 按记录读写文件

`size_t fread(void *ptr, size_t size, size_t nmemb, FILE *fp)`

`size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *fp)`

# 文件读写

■ **练习：** 从键盘上读取五条学生记录并将记录内容写入文件，然后从文件中读取信息并计算两门成绩的平均值，然后把学员信息输出到屏幕上

```
typedef struct{  
    int id;  
    char name[20];  
    unsigned math, eng;  
}std_t;
```

# 文件读写

## ■ 文件操作:

## ■ 读写

## ● 按格式读写文件

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

```
int fprintf(FILE *fp, const char *ptr, ...)
```

可方便实现变量向文件的输入、输出

# 文件读写

## ■ 文件操作:

## ■ 读写

## ● 字符串的输入输出

```
int sscanf(const char *str, const char *format, ...)
```

```
int sprintf(char *buf, const char *ptr, ...)
```

# 文件读写

## ■ 练习:

分别使用fprintf和sprintf实现printf

## Section 4



# 文件操作

---

## 文件定位

# 文件读写

## ■ 文件偏移：

■ 文件的读写操作都会影响文件偏移，文件偏移又叫文件的读写位置

# 文件读写

## ■ 文件操作：

### ■ 定位

`void rewind(FILE *fp)`: 文件的读写位置重新回到文件起始处

`long ftell(FILE *fp)`: 返回文件读写位置



# 文件读写

## ■ 文件操作：

### ■ 定位

`int fseek(FILE *fp, long offset, int whence)`

设置文件读写位置

Whence: `SEEK_SET`: 从文件起始处

`SEEK_CUR`: 当前读写位置

`SEEK_END`: 文件末尾

## Section 5

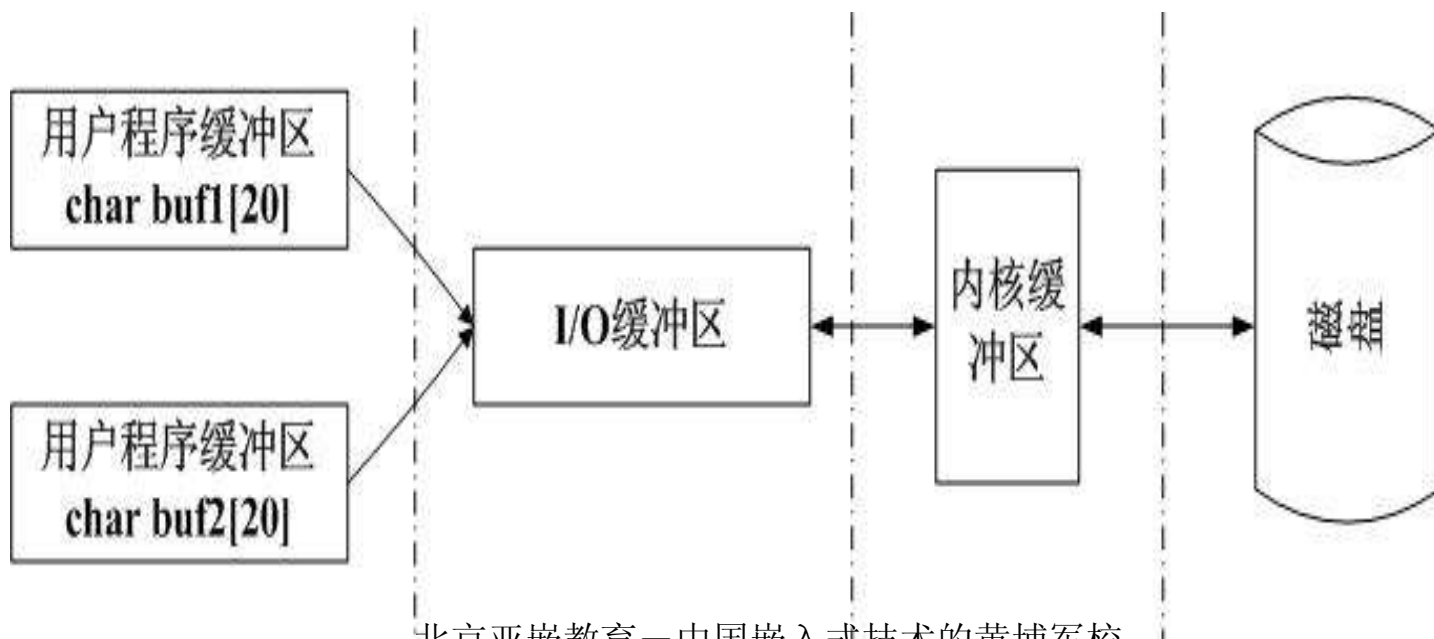


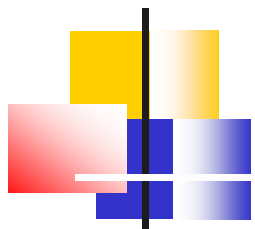
# 文件操作 缓冲区类别

---

# 缓冲区类别

■ 为了减少访问外设，提高执行效率，C标准库为每个打开的文件分配一个用户空间的I/O缓冲区。标准库中的文件读写函数都是从缓冲区中读写内容





# 缓冲区类别

## ■ I/O缓冲区缺点

- 程序向文件写入的内容不能直接反映到磁盘中

## ■ I/O缓冲区的分类

- 全缓冲：常规文件。I/O缓冲区满后写回内核
- 行缓冲：标准输入/输出设备文件。缓冲满或用户程序数据中有换行符时会写回内核
- 无缓冲：标准错误输出。用户程序的每次写操作都马上写入内核

# 缓冲区类别

- 清I/O缓冲的动作什么时候发生
  - 关闭文件（调用fclose）
  - 程序正常结束
  - 调用fflush函数： `int fflush(FILE *fp)`
    - 功能：冲刷fp指向文件的缓冲区
    - 参数：fp需要进行冲刷缓冲的文件的文件指针

# 文件操作

## ■ 课程练习

分别使用字节读写函数、字符串读写函数、按记录读写函数完成文件的复制

# 文件操作

## ■课程练习

编程读写一个文件test.txt，每个一秒向文件中写入一行记录，类似于这样：

1 2011-03-21 10:45:00

2 2011-03-21 10:45:01

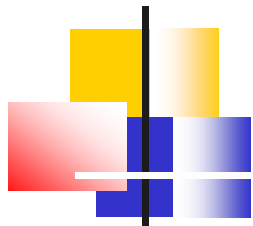
该程序应该无限循环，直到按ctrl-c终止，下次再启动程序应该在test.txt末尾追加记录并且序号能够接续上次的序号

# 文件操作

## ■ 课程练习

完成简单的学生成绩管理系统





# Let's DO it!

---

# Thanks for listening!

