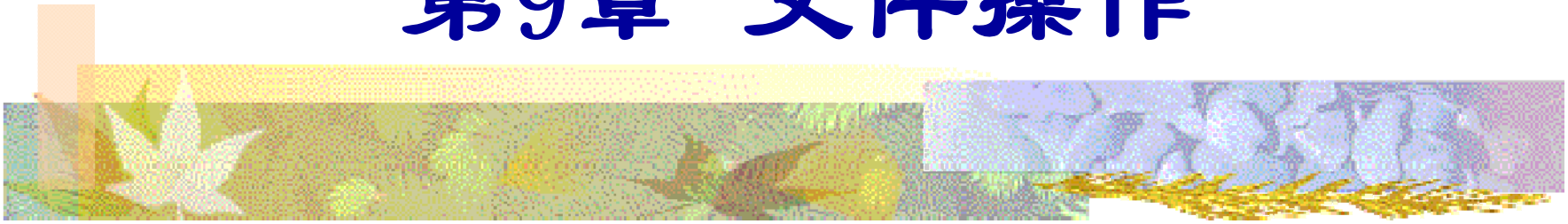
# 程序设计语言C

## 第9章 文件操作

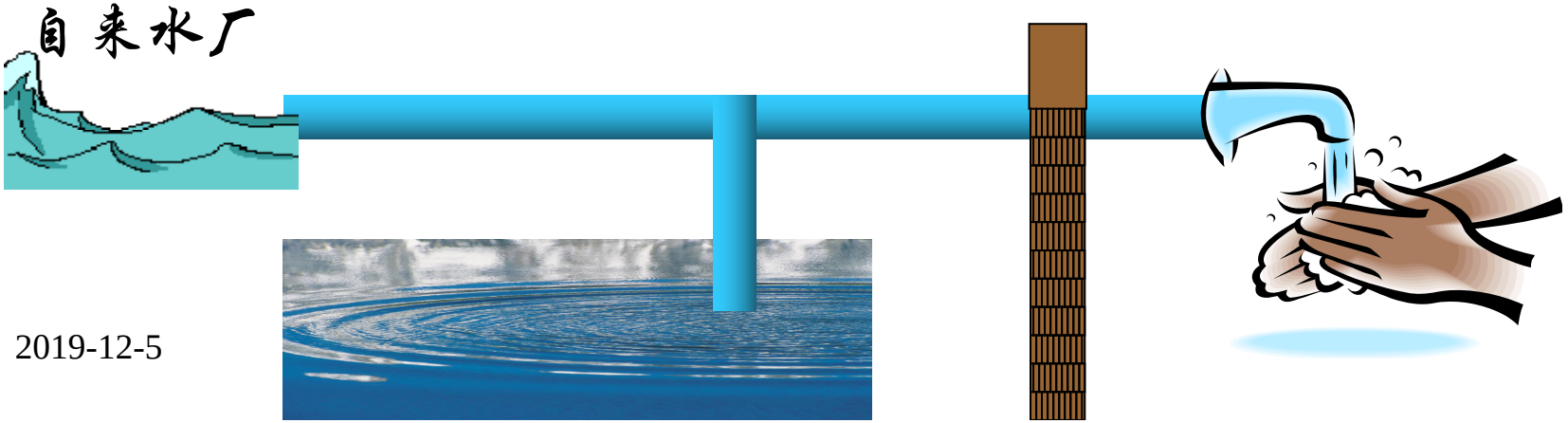


# I/O设备

- 输入设备
  - 键盘、鼠标
  - 软盘、硬盘（以文件的形式）
  - 串行口、并行口、USB接口、网络端口
  - 扫描仪、视频采集卡、电视卡、游戏杆、话筒
  - ……
- 输出设备
  - 显示器、打印机
  - 软盘、硬盘（以文件的形式）
  - 串行口、并行口、USB接口、网络端口
  - 音箱
  - ……
- 单纯的输入设备或者单纯的输出设备越来越少

# 标准输入输出

- 字符界面的操作系统一般都提供标准输入与输出设备
  - **DOS、Linux、Unix……**
- 一般情况，标准输入就是键盘，标准输出就是终端显示器
  - 操作系统有能力重定向标准输入与输出，比如让文件作为标准输入（标准输出）
  - 这种重定向程序本身是感觉不到的



# DOS下的标准输入输出重定向

- 程序prog如下
  - 
```
main()
{
    char c;
    while ((c=getchar()) != '\n')
        putchar(++c);
}
```
- 输入重定向
  - `prog < infile.txt`
- 输出重定向
  - `prog > outfile.txt`

# 流（Stream）

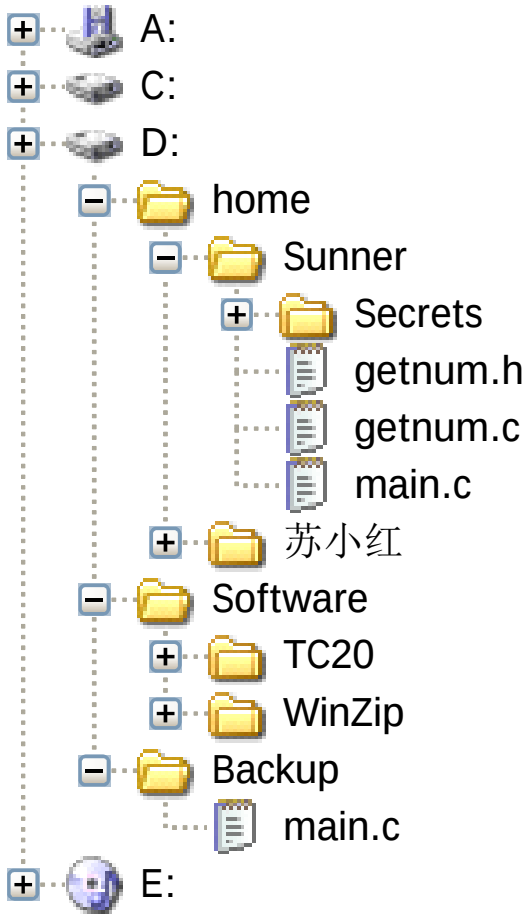
- 计算机中的流的概念
  - 一般称为数据流，也有叫做字节流、比特流的，还有很具体的文件流、视频流、音频流等
- 时光不能倒流，但计算机中的很多流都是会倒流的
  - 如果你想重新读已经读过的数据，或者要修改已经写入的数据，可以发出流控（**Flow Control**）命令
  - 不会倒流的数据流也很多，例如网络上的数据流。网络和数据线等介质只有很小的数据缓冲区，没有大量存储的能力

# 文件（File）的概念

- 计算机的内存容易健忘，所以数据必须保存在硬盘、软盘、光盘和磁带等“不健忘”的外存上
- 这些能大量、永久保存信息的媒介，一般都以文件的形式给用户及应用程序使用
- 文件
  - 一般指存储在外部介质上具有名字（文件名）的一组相关数据的集合
  - 用文件可长期保存数据，并实现数据共享
- 程序中的文件
  - 在程序运行时由程序在磁盘上建立一个文件，并通过写操作将数据存入该文件；
  - 或由程序打开磁盘上的某个已有文件，并通过读操作将文件中的数据读入内存供程序使用

# 文件的存放

- 可以建立若干目录（文件夹），在目录里保存文件，同一级目录里保存的文件不能同名。
- 对使用者而言，只要知道文件的路径（全目录）和文件名，就能使用该文件
  - **C:\home\Sunner\main.c**
  - 这都是托OS的福



# 文件的类型

- 二进制文件
  - 是一种字节序列，没有字符变换
  - 按照数据在内存中的存储形式存储到文件
  - 如整数127，在内存占2个字节，为0000000001111111，则文件中也存储为0000000001111111，占2个字节
- 文本文件/ASCII码文件
  - 是一种字符序列
  - 文件中存储每个字符的ASCII码
  - 如整数127在文件中占3个字节，分别存放这3个字符的ASCII码，即49，50，55

# 文件的格式

- 数据必须按照存入的类型读出，才能恢复其本来面貌
- 公开的标准格式
  - 如**bmp、tif、gif、jpg**和**mp3**等类型的文件，有大量软件能生成和使用这些类型的文件
- 也有不公开、甚至加密的文件格式
  - 如**Microsoft Word**的**doc**格式就不公开，所以至今还没有**Word**以外的其它软件能完美地读出**doc**文件

# C语言独特的文件访问

- 下面介绍的函数均定义在<stdio.h>中
- FILE *fopen(const char *filename, const char *mode);
  - FILE *fp = fopen("C:\\CONFIG.SYS", "rw");
- filename是文件名
  - 包含路径。如果不含路径，表示打开当前目录下的文件
- mode是打开方式
  - 常用为"r"、"w"、"rw"和"a"，分别表示只读、只写、读写和添加
  - "rb"表示只读二进制文件
- 返回值为指向此文件的指针，留待以后使用
  - 如果打开失败，返回值为NULL
- int fclose(FILE *fp);

# 文件指针（File Pointer）

- **FILE *fp ;**
  - 是FILE型指针变量
  - 标识一个特定的磁盘文件

```
typedef struct
{
    short level;              /*缓冲区‘满’或‘空’的程度*/
    unsigned flags;           /*文件状态标志*/
    char fd;                  /*文件描述符*/
    unsigned char hold;       /*如无缓冲区不读字符*/
    short bsize;              /*缓冲区的大小*/
    unsigned char *buffer;/*数据缓冲区的位置*/
    unsigned char *curp;      /*指针当前的指向*/
    unsigned istemp;          /*临时文件指示器*/
    short token;              /*用于有效性检查*/
}FILE;
```

- 在stdio.h文件中定义

# C语言独特的文件访问

- 字符读写
- `int fgetc(FILE *fp);`
  - 从**fp**读出一个字符并返回
  - 若读到文件尾，则返回**EOF**
- `int fputc(int c, FILE *fp);`
  - 向**fp**输出字符**c**
  - 若写入错误，则返回**EOF**，否则返回**c**

# C语言独特的文件访问

- 字符串读写
- `char *fgets(char *s,int n,FILE *fp);`

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

# C语言独特的文件访问

- 字符格式化读写
- 
```
int fscanf(FILE *fp,
           const char *format,
           ...);
```
  - `fscanf(fp, "%d,%6.2f", &i, &t);`
- 
```
int fprintf(FILE *fp,
            const char *format,
            ...);
```
  - `fprintf(fp, "%d,%6.2f", i, t);`

# C语言独特的文件访问

- 按数据块读写：
- 将 **ptr** 所指向的数组写到给定流**fp**中。
- **unsigned fwrite(const void *ptr,**
  **unsigned size,**
  **unsigned count,**
  **FILE *fp);**
  - **ptr:**指向带有尺寸 *size*count* 字节的内存块的指针
  - **size:**要写的每个元素的大小，以字节为单位
  - **count:**要写的元素/块的个数，每个元素的大小为 size 字节
  - **fp:**指向 FILE 对象的指针，该 FILE 对象指定了一个输入流
  - 返回实际写的数据块个数

# C语言独特的文件访问

- 按数据块读写
- 从给定流**fp**读取数据到 **ptr** 所指向的数组中。
- 
```
unsigned fread(void *ptr,
               unsigned size,
               unsigned count,
               FILE *fp);
```

  - **ptr:**指向带有最小尺寸 *size*count* 字节的内存块的指针
  - **size:**要读取的每个元素的大小，以字节为单位
  - **count:**最多读取的元素/块的个数，每个元素的大小为 size 字节
  - **fp:**指向 FILE 对象的指针，该 FILE 对象指定了一个输入流
  - 返回实际读到的数据块个数

# C语言独特的文件访问

- 文件定位，用于文件的随机读写
  - 打开的文件中有一个位置指针指示当前读写位置
  - 对文件每进行一次顺序读写，文件指针自动指向下一读写位置
- `int fseek(FILE *fp, long offset, int fromwhere);`
  - 把fp的位置指针从fromwhere开始移动offset个字节
  - fromwhere：SEEK_SET或0-----文件开始
  - SEEK_CUR或1-----当前位置
  - SEEK_END或2-----文件末尾
- `int ftell(FILE *fp);`
  - 返回fp的当前位置指针
- `int rewind(FILE *fp);`
  - 让fp的位置指针指向文件首字节

# C语言独特的文件访问

- 判断文件是否结束
- `int feof(FILE *fp);`
  - 当文件位置指针指向fp末尾时，返回非0值，否则返回0

# 错误处理

- 错误处理
  - 文件错误一般都是外界造成的，出错率很高
  - 被删除、修改、磁盘空间满、被其他文件打开
- 通过判断返回值发现错误
  - 所有文件操作出错时都返回-1
- 出错处理
  - 打印错误信息给用户，等待用户的处理
- `void perror(const char *s);`
  - 向标准错误输出字符串s，随后附上错误的文字说明

# 将int,double,float,char以字符流写

```
#include <stdio.h>  //按照字符方式写
int main(){
    int i = 100;
    double d = 3.14;
    float f = 0.01;
    char c = 'a';
    char buf[30] = "I love china!";
    FILE* fp = fopen("e:\\1.dat", "w+");
    fprintf(fp, "%d ", i);
    fprintf(fp, "%f ", d);
    fprintf(fp, "%f ", f);
    fprintf(fp, "%c ", c);
    fprintf(fp, "%s", buf);
    fclose(fp);
}
```

# 将int,double,float,char以字符流读

```
#include <stdio.h>  //按照字符方式写
int main(){
   //…
   fscanf(fp, "%d %lf %f %c", &i, &d, &f, &c);

   //比较fscanf(fp, “%s”, buf)    fgets(fp, buf);
   //fscanf(“%s”)以空格为分隔符,fgets读指定长度的字符串，不以空格为分隔符


}
```

# 将int,double,float,char以字节流写

```
#include <stdio.h>
int main(){
    int i = 100;
    double d = 3.14;
    float f = 0.01;
    char c = 'a';
    char buf[30] = "I love china!";
    FILE* fp = fopen("e:\\2.dat", "w");
    fwrite(&i, sizeof(i), 1, fp);
    fwrite(&d, sizeof(d), 1, fp);
    fwrite(&f, sizeof(f), 1, fp);
    fwrite(&c, sizeof(c), 1, fp);
    fwrite(buf, 30, 1, fp);
    fclose(fp);
}
```

# 将int,double,float,char以字符流读

```
#include <stdio.h>
int main(){
    int i;
    double d;
    float f;
    char c;
    char buf[30];
    FILE* fp = fopen("e:\\2.dat", "r");
    fread(&i, sizeof(i), 1, fp);
    fread(&d, sizeof(d), 1, fp);
    fread(&f, sizeof(f), 1, fp);
    fread(&c, sizeof(c), 1, fp);
    fread(buf, 30, 1, fp);
    fclose(fp);
    printf("%d %f %f %c %s", i, d, f,c, buf);
}
```

# 读写的一定要采用相同的流格式（字符/字节）

# 例1.文件复制

```
#include <stdio.h>

/*
文件复制:按照字符方式。
*/
int main(){
    FILE* pf1 = fopen("e:\\1.txt", "r");
    FILE* pf2 = fopen("e:\\a.txt", "w");
    char c = ' ';
    while((c=fgetc(pf1))!=EOF){
        fputc(c, pf2);
    }
    fflush(fp2); //强制缓冲区输出
    fclose(fp1); fclose(fp2);
}
```

# 文件复制

```
#include <stdio.h>
/*
文件复制:按照字节方式。
*/
int main(){
    FILE* pf1 = fopen("e:\\1.txt", "r");
    FILE* pf2 = fopen("e:\\a2.txt", "w");
    char buf[1024];
    int len = 0;
    while((len=fread(buf, 1, buf, pf1))!=0){
        fwrite(buf, len, fp2);
    }
    fflush(fp2); //强制缓冲区输出
    fclose(fp1); fclose(fp2);
}
```

```
#include <stdio.h>   //分析程序运行过程
struct Employee{
    char name[30];
    char code[10];
    float sal;
};
int main(){
    struct Employee e = {"liming", "1001", 8000};
    struct Employee e3;
    FILE* fp = fopen("e:\\1.dat", "wa");
    fwrite(&e, sizeof(e), 1, fp);//写完后，读写指针定位在文件最后
    fflush(fp);
    fseek(fp, 0, SEEK_SET);//指针定位到文件开始
    fread(&e3, sizeof(e), 1, fp);
    printf("%s", e.name);  //liming
    fclose(fp);
}
```

# 员工管理系统:数据的持久化存储

```
#include <stdio.h>
struct Employee{
//...
};
int main(){
    struct Employee es[N];
    int n = 0;
    //…

    //退出前，把员工信息写入文件
     FILE* fp2 = fopen("e:\\employee.dat", "w");
    //n为员工个数，假设所有员工顺序存储
    fwrite(es, sizeof(struct Employee), n, fp2);
    fclose(fp2);

}
```

# 员工管理系统:数据的持久化存储

```
#include <stdio.h>
struct Employee{
//...
};
int main(){
    struct Employee es[N];
    //启动时,从文件中读入员工信息
    struct Employee* p = NULL;
    p = es;
    FILE* fp = fopen("e:\\employee.dat", "r");
    int count = 0;
    int n = 0; //有多少员工
    while((count=fread(p, sizeof(struct Employee), 1, fp))==1){
        n++;
    }
}
```

# 员工管理系统:数据的持久化存储

```
#include <stdio.h>  //分析程序，如果按照如下方式存储，那么如何读出.
struct Employee{
//...
};
int main(){
    struct Employee es[N];
    int n = 0;
    //…
    //退出前，把员工信息写入文件
     FILE* fp2 = fopen("e:\\employee.dat", "w");
    //n为员工个数，假设所有员工顺序存储
    fwrite(&n, sizeof(int), 1, fp);
    fwrite(es, sizeof(struct Employee), n, fp2);
    fclose(fp2);
}
```

# 员工管理系统(链表存储）:数据的持久化存储

```
#include <stdio.h>
struct Employee{
//...
};
int main(){
    struct Employee* head;
    int n = 0;
    //退出前，把员工信息写入文件
    FILE* fp2 = fopen("e:\\employee.dat", "w");
    p = head;
    while(p){
       fwrite(p, sizeof(struct Employee), 1, fp2);
       p = p->next;
    }
    fclose(fp2);
}
```

# 员工管理系统(链表存储）:数据的持久化存储

```
#include <stdio.h>
struct Employee{
//...
};
int main(){
    struct Employee* head=NULL, *p=NULL;
    //启动时，把员工信息读入内存
    FILE* fp = fopen("e:\\employee.dat", "r");
    p = (struct Employee*)malloc(sizeof(struct Employee));
    //读入信息到p
    while((count=fread(p, sizeof(struct Employee), 1, fp))==1){
        p->next = head;  head = p;
        p = (struct Employee*)malloc(sizeof(struct Employee));
    }
    free(p); //释放最后一次申请的空间
    fclose(fp2);
}
```