1.标准IO

1.1snprintf/sprintf/fprintf函数

1.1.1c语言可变参数问题

```
#include <stdarg.h>

void va_start(va_list ap, last);
功能：根据last成员获取它的地址，存放到ap中（获取栈类型）
type va_arg(va_list ap, type);
功能：根据前面初始化号的ap，从last后取一个type（int/char/long）类型的数据
void va_end(va_list ap);
功能：销毁ap(释放ap占用的内存空间)
```

```
#include <head.h>
#include <stdarg.h>
int add(int n, ...)
{
    int i, sum = 0;
    va_list ap;
    //根据n初始化号一个ap成员
    va_start(ap, n);
    for (i = 0; i < n; i++) {
        //根据ap向后取int类型的数据加到sum中
        sum += va_arg(ap, int);
    }
    //销毁ap成员
    va_end(ap);

    return sum;
}
int main(int argc, const char* argv[])
{
    printf("sum = %d\n",add(2,100,200));
    printf("sum = %d\n",add(3,100,200,123));
    printf("sum = %d\n",add(5,100,200,300,400,500));
    return 0;
}
```

1.1.2sprintf/snprintf函数

```
int sprintf(char *str, const char *format, ...);
功能：将format控制格式中的字符串写入到str的数组中
参数：
    @str:存储格式化后字符串地址
    @format,...:控制格式，和printf一样
返回值：成功返回大于0的数，失败返回小于0的数

int snprintf(char *str, size_t size, const char *format, ...);
功能：将format控制格式中的字符串写入到str的数组中，最多size-1
参数：
    @str:存储格式化后字符串地址
    @size:大小
    @format,...:控制格式，和printf一样
返回值：成功返回大于0的数，失败返回小于0的数
```

```
#include <head.h>

int main(int argc,const char * argv[])
{
    char s[5];

    // 虽然控制格式中给的字符多于5个，但是实际
    // 只向s中写了4个字符，最后存的是'\0',不会有越界风险
    snprintf(s,sizeof(s),"abcdefghi");

    //sprintf没有填写大小的成员，所以它会有越界的风险，
    //如果越界就会有内存访问错误问题,所以以后优先选择snprintf
    //sprintf(s,"abcdefghi");
    printf("s = %s\n",s);
    return 0;
}
```

1.1.3fprintf函数

```
int fprintf(FILE *stream, const char *format, ...);
功能：将format格式化后的字符串写入到文件中
参数：
    @stream:文件指针
    @format,...控制格式
返回值：成功返回正数，失败返回负数
```

```
用法1：
fprintf(fp, "%d.%d-%02d-%02d %02d:%02d:%02d\n",
       line++, tm->tm_year + 1900, tm->tm_mon + 1, tm->tm_mday,
       tm->tm_hour, tm->tm_min, tm->tm_sec);
用法2：
if (argc != 3) {
    fprintf(stderr,"input error,try again\n");
    fprintf(stderr,"usage: ./a.out srcfile destfile\n");
    return -1;
}
```

1.2fseek函数

1.2.1fseek函数的功能

```
int fseek(FILE *stream, long offset, int whence);
功能：修改文件中光标位置
参数：
    @stream:文件指针
    @offset:偏移
        >0 :向后偏移
        =0 :不偏移
        <0 :向前偏移
    @whence:从那个位置偏移
        SEEK_SET ： 开头
        SEEK_CUR ： 当前位置
        SEEK_END ： 结尾
返回值：成功返回0，失败返回-1置位错误码
```

1.2.2fseek函数的实例

```
#include <head.h>

int main(int argc, const char* argv[])
{
    FILE* fp;
```

```c
    if ((fp = fopen("./hello.txt", "r+")) == NULL)
        PRINT_ERR("fopen error");

    //将光标从文件开头向后偏移5个字节
    if(fseek(fp,5,SEEK_SET))
        PRINT_ERR("fseek error");

    printf("->%c\n",fgetc(fp));
    fputc('K',fp);

    fclose(fp);
    return 0;
}
```

1.2.3fseek函数的练习

练习：使用a+的方式打开文件，使用fseek修改光标，看能否影响读和写的位置。

对于a+方式打开的文件，可以修改读光标位置，写永远都在结尾，修改光标没有用。

```c
#include <head.h>

int main(int argc, const char* argv[])
{
    FILE* fp;
    if ((fp = fopen("./hello.txt", "a+")) == NULL)
        PRINT_ERR("fopen error");

    if(fseek(fp,5,SEEK_SET))
        PRINT_ERR("fseek error");
    fputc('K',fp); //在结尾写

    if(fseek(fp,5,SEEK_SET))
        PRINT_ERR("fseek error");
    printf("%c\n",fgetc(fp));//读的是第6个字符

    fclose(fp);
    return 0;
}
```

1.3rewind函数

```
void rewind(FILE *stream);
功能：将光标恢复到文件的开头 （等价于 = fseek(fp,0,SEEK_SET)）
参数：
    @stream:文件指针
返回值：无
```

1.4ftell函数

1.4.1ftell函数的功能

```
long ftell(FILE *stream);
功能：返回光标到文件开头的字节数
参数：
    @stream:文件指针
返回值：成功返回光标到文件开头的字节数，失败返回-1置位错误码
```

1.4.2ftell函数的实例

```c
#include <head.h>

int main(int argc, const char* argv[])
{
    FILE* fp;
    long size;
    if (argc != 2) {
        fprintf(stderr, "input error,try again\n");
        fprintf(stderr, "usage:./a.out filename\n");
        return -1;
    }
    if ((fp = fopen(argv[1], "r")) == NULL)
        PRINT_ERR("fopen error");

    fseek(fp, 0, SEEK_END); // 将光标定位到文件结尾

    if ((size = ftell(fp)) == -1)
        PRINT_ERR("ftell error");

    printf("file size = %ld\n", size);

```

```
    fclose(fp);
    return 0;
}
```

1.5使用标准IO操作图片

https://blog.csdn.net/qq_41137110/article/details/119893817

1.5.1bmp图片组成

1. **bmp文件头(bmp file header)**：提供文件的格式、大小等信息（14字节）

变量名	地址偏移	大小	作用
bfType	0000h	2 bytes	说明文件的类型，可取值为： • BM – Windows 3.1x, 95, NT, ... • BA – OS/2 Bitmap Array • CI – OS/2 Color Icon • CP – OS/2 Color Pointer • IC – OS/2 Icon • PT – OS/2 Pointer
bfSize	0002h	4 bytes	说明该位图文件的大小，用字节为单位
bfReserved1	0006h	2 bytes	保留，必须设置为0
bfReserved2	0008h	2 bytes	保留，必须设置为0
bfOffBits	000Ah	4 bytes	说明从文件头开始到实际的图象数据之间的字节的偏移量。 这个参数是非常有用的，因为位图信息头和调色板的长度会根据不同情况而变化， 所以我们可以用这个偏移值迅速的从文件中读取到位图数据。

2. **位图信息头(bitmap information)**：提供图像数据的尺寸、位平面数、

   压缩方式、颜色索引等信息（40字节）

变量名	地址偏移	大小	作用
biSize	000Eh	4 bytes	BITMAPINFOHEADER结构所需要的字数。
biWidth	0012h	4 bytes	说明图像的宽度，用像素为单位
biHeight	0016h	4 bytes	说明图像的高度，以像素为单位。 注：这个值除了用于描述图像的高度之外，它还有另一个用处，就是指明该图像是倒向的位图，还是正向的位图。 如果该值是一个正数，说明图像是倒向的，如果该值是一个负数，则说明图像是正向的。 大多数的BMP文件都是倒向的位图，也就是高度值是一个正数。
biPlanes	001Ah	2 bytes	为目标设备说明颜色平面数， 其值将总是被设为1。
biBitCount	001Ch	2 bytes	说明比特数/像素，其值为1、4、8、16、24或32。
biCompression	001Eh	4 bytes	说明图像数据压缩的类型。取值范围： 0 BI_RGB 不压缩（最常用） 1 BI_RLE8 8比特游程编码（RLE），只用于8位位图 2 BI_RLE4 4比特游程编码（RLE），只用于4位位图 3 BI_BITFIELDS 比特域，用于16/32位位图 4 BI_JPEG JPEG 位图含JPEG图像（仅用于打印机） 5 BI_PNG PNG 位图含PNG图像（仅用于打印机）
biSizeImage	0022h	4 bytes	说明图像的大小， 以字节为单位。当用BI_RGB格式时，可设置为0。
biXPelsPerMeter	0026h	4 bytes	说明水平分辨率，用像素/米表示，有符号整数
biYPelsPerMeter	002Ah	4 bytes	说明垂直分辨率，用像素/米表示，有符号整数
biClrUsed	002Eh	4 bytes	说明位图实际使用的彩色表中的颜色索引数 （设为0的话，则说明使用所有调色板项）
biClrImportant	0032h	4 bytes	说明对图像显示有重要影响的颜色索引的数目 如果是0，表示都重要。

3. **位图数据(bitmap data)**：就是图像数据

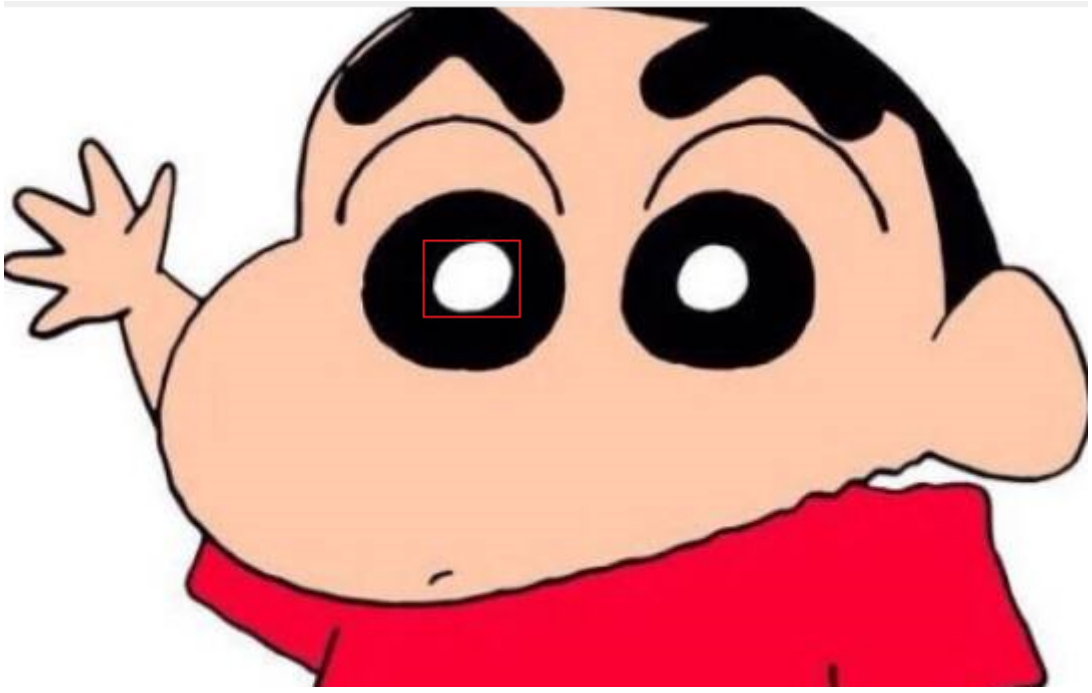
   RGB888 一个像素点占3字节

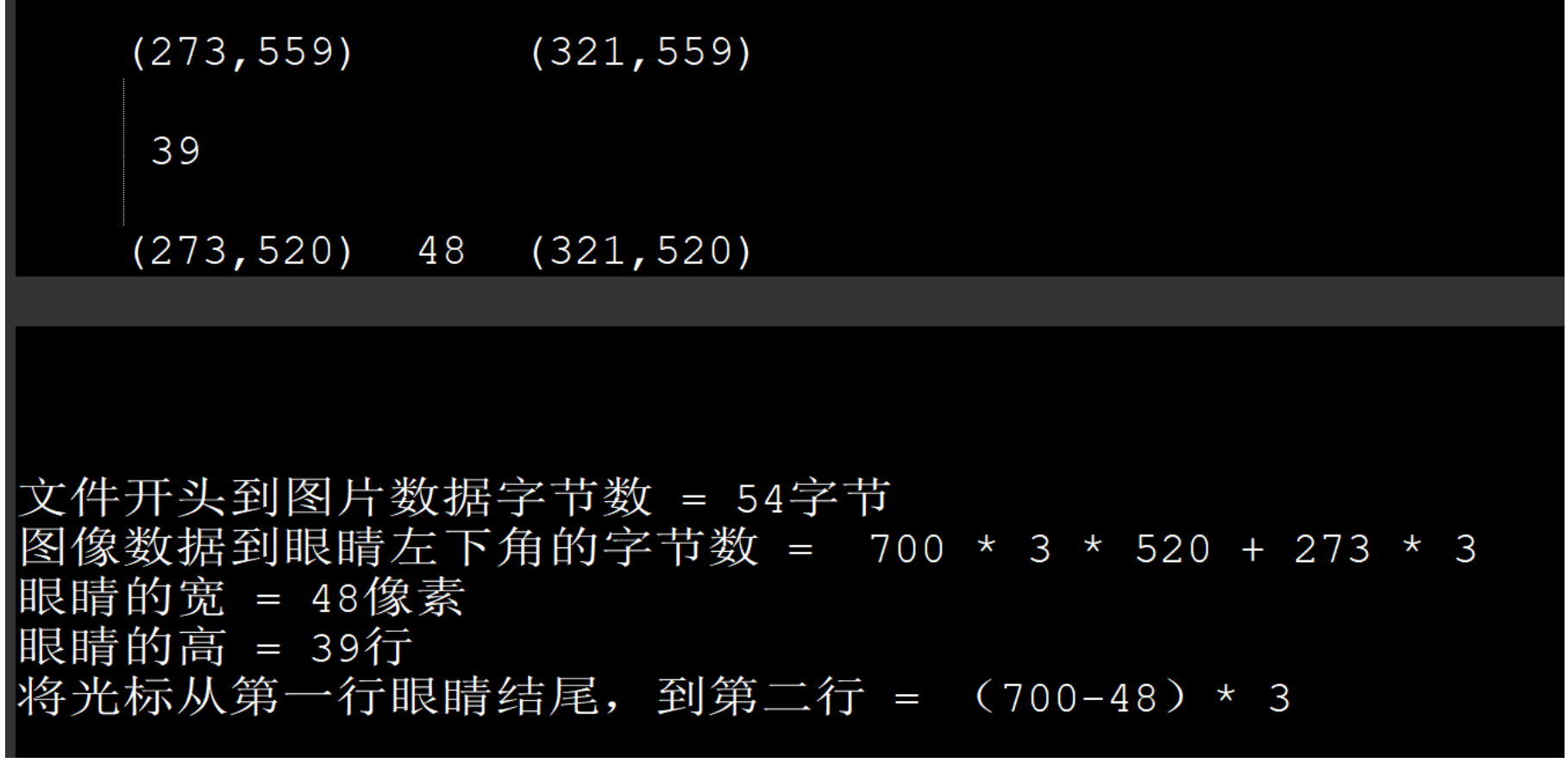
   RGB565 一个像素点占2字节

1.5.2使用标准IO操作图片实例

```c
#include <head.h>

typedef struct {
    unsigned char B;
    unsigned char G;
    unsigned char R;
} RGB_t;

// ./a.out xxx.bmp
int main(int argc, const char* argv[])
{
    FILE* fp;
    unsigned int size, offset, width, high;
    unsigned short bitcount;
    if (argc != 2) {
        fprintf(stderr, "input error,try again\n");
        fprintf(stderr, "usage: ./a.out xxx.bmp\n");
        return -1;
    }

    if ((fp = fopen(argv[1], "r+")) == NULL)
        PRINT_ERR("fopen error");

    fseek(fp, 2, SEEK_SET);
    fread(&size, 4, 1, fp);
    printf("size = %d\n", size);

    fseek(fp, 10, SEEK_SET);
    fread(&offset, 4, 1, fp);
    printf("offset = %d\n", offset);

    fseek(fp, 18, SEEK_SET);
    fread(&width, 4, 1, fp);
    printf("width = %d\n", width);

    fread(&high, 4, 1, fp);
    printf("high = %d\n", high);

    fseek(fp, 2, SEEK_CUR);
    fread(&bitcount, 2, 1, fp);
    printf("bitcount = %d\n", bitcount);

    fseek(fp, 54, SEEK_SET);

    RGB_t rgb = { 0, 0, 0xff };
    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < width; j++) {
            fwrite(&rgb, sizeof(rgb), 1, fp);
        }
    }
    return 0;
}
```

1.5.3给图片局部着色






```c
#include <head.h>

typedef struct {
    unsigned char B;
    unsigned char G;
    unsigned char R;
} RGB_t;

// ./a.out xxx.bmp
int main(int argc, const char* argv[])
{
    FILE* fp;
    unsigned int size, offset, width, high;
    unsigned short bitcount;
    if (argc != 2) {
        fprintf(stderr, "input error,try again\n");
        fprintf(stderr, "usage: ./a.out xxx.bmp\n");
        return -1;
    }

    if ((fp = fopen(argv[1], "r+")) == NULL)
        PRINT_ERR("fopen error");

    fseek(fp, 2, SEEK_SET);
    fread(&size, 4, 1, fp);
    printf("size = %d\n", size);

    fseek(fp, 10, SEEK_SET);
    fread(&offset, 4, 1, fp);
    printf("offset = %d\n", offset);

    fseek(fp, 18, SEEK_SET);
    fread(&width, 4, 1, fp);
    printf("width = %d\n", width);

    fread(&high, 4, 1, fp);
    printf("high = %d\n", high);

    fseek(fp, 2, SEEK_CUR);
    fread(&bitcount, 2, 1, fp);
    printf("bitcount = %d\n", bitcount);

    fseek(fp, offset+width*3*520+273*3, SEEK_SET);

    RGB_t rgb = { 0, 0, 0xff };
    for (int i = 0; i < 39; i++) {
        for (int j = 0; j < 48; j++) {
            fwrite(&rgb, sizeof(rgb), 1, fp);
        }
        fseek(fp,(width-48)*3,SEEK_CUR);
    }
    return 0;
}
```

2.文件IO

2.1什么是fd

fd:文件描述符，它是一个整数。当使用open打开文件的时候就会产生fd。

这个文件描述符就代表打开的文件。以后在使用read/write/close操作文件

都是通过fd完成的。在一个正在运行的程序中默认0,1,2这三个文件描述符

已经被占用了，分别对应的是标准输入，标准输出，标准出错。在Ubuntu系统

中默认配置情况一个整型最多有1024个文件描述符，用户最多可以打开1021个

文件。可以通过ulimit -a查看，可以通过`ulimit -n 数值`修改打开文件的个数。

类别	标准IO	文件IO
标准输入	stdin(stdin->_fileno)	0
标准输出	stdout(stdout->_fileno)	1
标准出错	stderr(stderr->_fileno)	2

2.2open函数

2.2.1open函数的功能

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
功能：使用文件IO方式打开文件
参数：
    @pathname:想要打开文件的路径及名字
    @flags:打开文件的方式
        O_RDONLY：以只读方式打开
        O_WRONLY：以只写方式打开
        O_RDWR  ：以读写方式打开
        O_APPEND：以追加的方式打开
        O_CREAT：创建文件，如果第二个参数中有O_CREAT,必须填充第三个参数
        O_TRUNC ：清空文件
        O_EXCL ：需要和O_CREAT结合使用，如果文件不存在就创建，如果文件存在
                会返回已存在的错误码EEXIST
    @mode:创建文件的权限 （0666）
        文件被从创建出来的权限 = (mode & ~umask) = （0666 & 0664） = 0664
        umask是文件的掩码，可以通过umask命令查看，也可以通过umask命令修改
        umask的默认值是0002，普通文件的最大权限是0666。所以在最大文件权限基础上取反
        ~0002 = ~000 000 010 = 111 111 101 = 0775   （在0777基础上取反的）
        ~0002 = ~000 000 010 = 110 110 100 = 0664   （在0666基础上取反的）
返回值：成功返回新的文件描述符，失败返回-1置位错误码
```

2.2.2标准IO和文件IO打开方式对比

文件IO	标准IO	功能
O_RDONLY	"r"	以只读的方式打开文件，光标在文件的开头
O_RDWR	"r+"	以读写方式打开文件，光标定位到文件的开头
O_WRONLY|O_CREAT|O_TRUNC,0666	"w"	以只写方式打开文件，文件不存在会创建，文件存在清空，光标在开头
O_RDWR|O_CREAT|O_TRUNC,0666	"w+"	以读写方式打开文件，文件不存在会创建，文件存在清空，光标在开头
O_WRONLY|O_APPEND|O_CREAT,0666	"a"	以追加的方式打开文件，文件不存在就创建，光标在文件的结尾
O_RDWR|O_APPEND|O_CREAT,0666	"a+"	以读和追加的方式打开文件，文件不存在就创建，读光标在开头，写光标在结尾

2.2.3open函数的实例

类似于标准IO w+ 打开文件

```
#include <head.h>

int main(int argc, const char* argv[])
{
    int fd;
    //类似于标准IO w+ 打开文件
    if ((fd = open("./hello.txt", O_RDWR | O_CREAT | O_TRUNC,0664)) == -1)
        PRINT_ERR("open error");

    return 0;
}
```

如果文件存在以只读方式打开，如果文件不存在创建并以只写方式打开

```c
#include <head.h>

int main(int argc, const char* argv[])
{
    int fd;
    //如果文件存在以只读方式打开，如果文件不存在创建并以只写方式打开
    if ((fd = open("./hello.txt", O_WRONLY | O_CREAT | O_EXCL ,0664)) == -1){
        if(errno == EEXIST){
            fd = open("./hello.txt", O_RDONLY);
            printf("文件已存在，以只读方式打开,fd = %d\n",fd);
        }else{
            PRINT_ERR("open error");
        }
    }else{
        printf("文件不存在，创建成功了，并以只写方式打开,fd = %d\n",fd);
    }
    return 0;
}
```

2.3read函数

2.3.1read函数的功能

```c
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
功能：从fd对应文件中，将数据读到buf中，读的大小是count(字节)
参数：
    @fd:文件描述符
    @buf:存储读取到数据的首地址
    @count:想要读的字节数
返回值：成功返回读取到的字节数，如果返回0代表读取到了文件的结尾
        失败返回-1，置位错误码。
```

2.3.2read函数的实例

```c
#include <head.h>

int main(int argc, const char* argv[])
{
    int fd, ret;
    if ((fd = open("./hello.txt", O_RDONLY)) == -1)
        PRINT_ERR("open error");

    char s[11] = { 0 };
    //read函数成功返回大于0的值，等于0或者小于0就代表到结尾或者失败
    //如果读到了数据循环继续，否则退出循环。
    while ((ret = read(fd, s, 10)) > 0) {
        //将read的函数中打印到终端上
        printf("ret = %d\n", ret);
        //将读取到的字符串打印到终端上（s多定义一个，为了printf访问的时候有'\0'）
        printf("s = %s", s);
        //防止对第二次read有影响
        memset(s, 0, sizeof(s));
    }

    return 0;
}
```

2.4write函数

2.4.1write函数的功能

```c
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
功能:将buf中的数据写入到文件中，写的大小是count
参数:
    @fd:文件描述符
    @buf:被写数据的首地址
    @count:想要写的字节数
返回值：成功返回写的字节数，失败返回-1置位错误码
```

2.4.2write函数的实例

```c
#include <head.h>
typedef struct{
    char name[20];
    char sex;
    int age;
}stu_t;
```

```c
int main(int argc, const char* argv[])
{
    int fd, ret;
    if ((fd = open("./hello.txt", O_RDWR | O_CREAT | O_TRUNC, 0666)) == -1)
        PRINT_ERR("open error");

    // char s[] = "hello everyone~~~";
    // write(fd,s,strlen(s));

    // int num=12345;
    // write(fd,&num,4);

    stu_t stu = {"zhangsan",'m',22};
    write(fd,&stu,sizeof(stu));
    return 0;
}
```

2.4.3write/read函数的练习

练习：使用read/write函数实现文件的拷贝

./a.out srcfile destfile

```c
#include <head.h>

int main(int argc, const char* argv[])
{
    int fd1, fd2, ret;
    char s[30] = { 0 };
    if (argc != 3) {
        fprintf(stderr, "input error,try again\n");
        fprintf(stderr, "usage: ./a.out srcfile destfile\n");
        return -1;
    }

    if ((fd1 = open(argv[1], O_RDONLY)) == -1)
        PRINT_ERR("open src error");

    if ((fd2 = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0666)) == -1)
        PRINT_ERR("open dest error");

    while ((ret = read(fd1, s, sizeof(s))) > 0) {
        write(fd2,s,ret);
    }

    close(fd1);
    close(fd2);
    return 0;
}
```

2.5close函数

2.5.1close函数的功能

```
#include <unistd.h>
int close(int fd);
功能：关闭文件
参数：
    @fd:文件描述符
返回值：成功返回0，失败返回-1置位错误码
```

2.5.2close函数实例

```c
#include <head.h>

int main(int argc, const char* argv[])
{
    int fd;
    printf("1111111111\n");  //printf->stdout->1文件描述符
    close(1);                // 关闭标准输出
    printf("2222222222\n");  //因为上述已经将1文件描述符关闭了，所以这句换不会在终端显示了

    //open在打开文件的时候，文件描述符采用最小未使用的原则分配，这里分配的fd=1
    if ((fd = open("./hello.txt", O_RDWR | O_CREAT | O_TRUNC, 0666)) == -1)
        PRINT_ERR("open error");

    // 问：以下语句printf的结果为什么在hello.txt文件中
    printf("3333333333\n"); //printf->stdout->1文件描述符

    return 0;
}
```

2.6lseek函数

2.6.1lseek函数的功能

```
#include <sys/types.h>
#include <unistd.h>

off_t lseek(int fd, off_t offset, int whence);
功能：修改光标的位置
参数：
    @fd:文件描述符
    @offset:偏移
        >0 :向后偏移
        =0 :不偏移
        <0 :向前偏移
    @whence:从哪偏移
        SEEK_SET:开头
        SEEK_CUR:当前位置
        SEEK_END:结尾
返回值：成功返回光标到文件开头的字节数，失败返回-1置位错误码
```

2.6.2lseek函数的实例

```
#include <head.h>

int main(int argc, const char* argv[])
{
    int fd;
    long size;
    if (argc != 2) {
        fprintf(stderr, "input error,try again\n");
        fprintf(stderr, "usage:./a.out filename\n");
        return -1;
    }

    if ((fd = open(argv[1], O_RDONLY)) == -1)
        PRINT_ERR("fopen error");

    if ((size = lseek(fd,0,SEEK_END)) == -1)
        PRINT_ERR("lseek error");

    printf("file size = %ld\n", size);

    close(fd);
    return 0;
}
```

3.作业

1. 使用文件IO将图片局部着色