Примеры реализации некоторых библиотечных функций (K&R)

```
#define BUFSIZ  1024
main( ) { /* копирование ввода на вывод */
    char buf [BUFSIZ];
    int n;
    while ((n = read(0, buf, BUFSIZ)) > 0)
           write(1,buf,n);
    return 0;
}
```

```
/* getchar: небуферизованный ввод одной литеры */
int getchar(void) {
  char c;
  return (read(0, &c, 1) == 1) ? (unsigned char) c : EOF;
}
```

```
 /* getchar: простая версия с буферизацией */
int getchar(void) {
    static char buf[BUFSIZ];
    static char *bufp = buf;
    static int n = 0;
    if (n == 0) { /*- буфер пуст */
        n = read(0, buf, sizeof buf);
        bufp = buf;
         }
      return (--n >= 0) ? (unsigned char) *bufp++ : EOF;
}
```

Структура данных FILE для обработки открытых потоков и набор функций, связанных с ней

```
#define NULL      0
#define EOF       (-1)
#define BUFSIZ    1024
#define OPEN_MAX 20   /* макс. число одновр. открытых файлов*/
typedef struct _iobuf {
int cnt;       /* колич. неиспользованных позиций в буфере */
char ptr;      /* позиция следующей литеры */
char base;     /* адрес буфера */
int flag;      /* режим доступа */
int fd;        /* дескриптор файла */
} FILE;
```

```
enum _flag {
      _READ =01,       / *файл открыт на чтение */
      _WRITE = 02,     /* файл открыт на запись */
      _UNBUF =04,      /* файл не буферизуется */
      _EOF   = 010,    /* в данном файле «наступил EOF» */
      _ERR   = 020     /* в данном файле «случилась ошибка»*/
}

FILE _iob[OPEN_MAX] = {     /* stdin, stdout, stderr: */
 { 0, (char *) 0, (char *) 0, _READ,           0 },
 { 0, (char *) 0, (char *) 0, _WRITE,          1 },
 { 0, (char *) 0, (char *) 0, _WRITE | _UNBUF, 2 }
}

#define stdin  (&_iob[0])
#define stdout (&_iob[1])
#define stderr (&_iob[2])


/* _fillbuf : запрос памяти и заполнение буфера */
int _fillbuf(FILE *fp)
{
  int bufsize;
  if ((fp->flag&(_READ|_EOF|_ERR)) != _READ)
    return EOF;
  bufsize = (fp->flag & _UNBUF) ? 1 : BUFSIZ;
  if (fp->base == NULL)   /* буфера еще нет */
      if ((fp->base = (char *) malloc(bufsize)) == NULL)
          return EOF;     /* нельзя получить буфер */
  fp->ptr = fp->base;
  fp->cnt = read(fp->fd, fp->ptr, bufsize);
  if (--fp->cnt < 0) {
      if (fp->cnt == -1)
           fp->flag |= _EOF;
      else
          fp->flag |= _ERR;
      fp->cnt = 0;
  return EOF;
  }
  return (unsigned char) *fp->ptr++;
}

#define feof(p)     (((p)->flag & _EOF) != 0)
#define ferror(p)   (((p)->flag & _ERR) != 0)
#define fileno(p)   ((p)->fd)
#define getc(p)   (     --(p)->cnt >= 0 \
                     ? (unsigned char) *(p)->ptr++ \
                     : _fillbuf(p) \
                  )
```

```
/* _flushbuf : запись заполненного буфера в файл */
int _flush buf(char sym, FILE *fp){
/*
    Записывает с помощью системного вызова write содержимое
буфера fp->buf   и символ sym в файл с дескриптором fp->fd.
Возвращает записанный символ sym. Реализоовать самостоятельно.
*/

#define putc(x,p) (     --(p)->cnt >= 0 \
                        ? *(p)->ptr++ = (x)\
                        : _flushbuf((x),p) \
                  )

#define getchar()   getc(stdin)
#define putchar(x) putc((x), stdout)


/* реализация fopen ( ) */

#define PERMS 0666 /* RW для собственника, группы и остальных */

/* fopen: открывает файл, возвращает файловый указатель */
FILE  *fopen(char *name, char *mode)
{
    int fd;
    FILE *fp;
    if (*mode != 'r' && *mode != 'w' && *mode != 'a')
    return NULL;
    for (fp = _iob; fp < _iob + OPEN_MAX; fp++)
            if ((fp->flag & (_READ | _WRITE)) == 0)
    break;     /* найдено своб.место для описателя */
     if (fp >= _iob + OPENMAX)  /* нет места для описателя */
        return NULL;
     if (*mode == 'w')
         fd = creat(name, PERMS);
     else if (*mode == 'a') {
         if ((fd = open(name, O_WRONLY, 0)) == -1)
               fd = creat(name, PERMS);
            lseek(fd, 0L, 2);
          } else
                fd = open(name, O_RDONLY, 0);
     if (fd == -1)   /* не возможен доступ по имени name */
    return NULL;
     fp->fd = fd;
     fp->cnt = 0;
     fp->base = NULL;
     fp->flag = (*mode == 'r') ? _READ: _WRITE;
     return fp;
}
```

/* Самостоятельно реализовать int fclose(FILE *fp)

fclose производит дозапись еще не записанных буферизованных данных, сбрасывает несчитанный буферизованный ввод, освобождает память из-под всех автоматически запрошенных буферов, после чего закрывает файл с помощью системного вызова close(fp->fd). Сбрасывет все флаги fp->flags в ноль, делая тем самым структуру *fp свободной для использования функцией fopen().

Возвращает EOF в случае ошибки и нуль в противном случае.

* /

/* Самостоятельно реализовать int fflush(FILE *fp)

Применяемая к потоку вывода функция fflush производит дозапись всех оставшихся в буфере (еще не записанных) данных; для потока ввода эта функция не определена. Возвращает EOF в случае возникшей при записи ошибки или нуль в противном случае. Обращение вида fflush(NULL) выполняет указанные операции для всех потоков вывода.

* /