Функции fread() и fwrite()

Для чтения и записи данных, тип которых может занимать более 1 байта, в файловой системе языка С имеется две функции: fread() и fwrite(). Эти функции позволяют читать и записывать блоки данных любого типа. Их прототипы следующие:

```
size_t fread(void *буфер, size_t колич_байт, size_t счетчик, FILE *уф);
size_t fwrite(const void *буфер, size_t колич_байт, size_t счетчик, FILE *уф);
```

Для fread() *буфер* – это указатель на область памяти, в которую будут прочитаны данные из файла. А для fwrite() *буфер* – это указатель на данные, которые будут записаны в файл. Значение *счетчик* определяет, сколько считывается или записывается элементов данных, причем длина каждого элемента в байтах равна *колич_байт*. (Вспомните, что тип size_t определяется как одна из разновидностей целого типа без знака.) И, наконец, *уф* – это указатель файла, то есть на уже открытый поток.

Функция fread() возвращает количество прочитанных элементов. Если достигнут конец файла или произошла ошибка, то возвращаемое значение может быть меньше, чем счетчик. А функция fwrite() возвращает количество записанных элементов. Если ошибка не произошла, то возвращаемый результат будет равен значению счетчик.

Использование fread() и fwrite()

Как только файл открыт для работы с двоичными данными, fread() и fwrite() соответственно могут читать и записывать информацию любого типа. Например, следующая программа записывает в дисковый файл данные типов double, int и long, а затем читает эти данные из того же файла. Обратите внимание, как в этой программе при определении длины каждого типа данных используется функция sizeof().

```
/* Запись несимвольных данных в дисковый файл
   и последующее их чтение.  */
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  FILE *fp;
  double d = 12.23;
  int i = 101;
  long l = 123023L;

  if((fp=fopen("test", "wb+"))==NULL) {
    printf("Ошибка при открытии файла.\n");
    exit(1);
  }

  fwrite(&d, sizeof(double), 1, fp);
  fwrite(&i, sizeof(int), 1, fp);
  fwrite(&l, sizeof(long), 1, fp);

  rewind(fp);

  fread(&d, sizeof(double), 1, fp);
  fread(&i, sizeof(int), 1, fp);
  fread(&l, sizeof(long), 1, fp);

  printf("%f %d %ld", d, i, l);

  fclose(fp);

  return 0;
}
```

Как видно из этой программы, в качестве буфера можно использовать (и часто именно так и делают) просто память, в которой размещена переменная. В этой простой программе значения, возвращаемые функциями fread() и fwrite(), игнорируются. Однако на практике эти значения необходимо проверять, чтобы обнаружить ошибки.

Одним из самых полезных применений функций fread() и fwrite() является чтение и запись данных пользовательских типов, особенно структур. Например, если определена структура

```
struct struct_type {
  float balance;
  char name[80];
} cust;
```

то следующий оператор записывает содержимое cust в файл, на который указывает fp:

```
fwrite(&cust, sizeof(struct struct_type), 1, fp);
```

Пример со списком рассылки

Чтобы показать, как можно легко записывать большие объемы данных, пользуясь функциями fread() и fwrite(), мы переделаем программу работы со списком рассылки, с которой впервые встретились в главе 7. Усовершенствованная версия сможет сохранять адреса в файле. Как и раньше, адреса будут храниться в массиве структур следующего типа:

```
struct addr {
  char name[30];
  char street[40];
  char city[20];
  char state[3];
  unsigned long int zip;
} addr_list[MAX];
```

Значение MAX определяет максимальное количество адресов, которое может быть в списке.

При выполнении программы поле name каждой структуры инициализируется пустым указателем (NULL). В программе свободной считается та структура, поле name которой содержит строку нулевой длины, т.е. имя адресата представляет собой пустую строку.

Далее приведены функции save() и load(), которые используются соответственно для сохранения и загрузки базы данных (списка рассылки). Обратите внимание, насколько кратко удалось закодировать каждую из функций, а ведь эта краткость достигнута благодаря мощи fread() и fwrite()! И еше обратите внимание на то, как эти функции проверяют значения, возвращаемые функциями fread() и fwrite(), чтобы обнаружить таким образом возможные ошибки.

```
/* Сохранение списка. */
void save(void)
{
  FILE  *fp;
  register int i;

  if((fp=fopen("maillist", "wb"))==NULL) {
    printf("Ошибка при открытии файла.\n");
    return;
  }

  for(i=0; i<MAX; i++)
    if(*addr_list[i].name)
      if(fwrite(&addr_list[i],
         sizeof(struct addr), 1, fp)!=1)
           printf("Ошибка при записи файла.\n");

  fclose(fp);
}

/* Загрузить файл. */
void load(void)
{
  FILE  *fp;
  register int i;

  if((fp=fopen("maillist", "rb"))==NULL) {
    printf("Ошибка при открытии файла.\n");
    return;
  }
```

```
  init_list();
  for(i=0; i<MAX; i++)
    if(fread(&addr_list[i],
       sizeof(struct addr), 1, fp)!=1) {
         if(feof(fp)) break;
         printf("Ошибка при чтении файла.\n");
    }

  fclose(fp);
}
```

Обе функции, `save()` и `load()`, подтверждают (или не подтверждают) успешность выполнения функциями `fread()` и `fwrite()` операций с файлом, проверяя значения, возвращаемые функциями `fread()` и `fwrite()`. Кроме того, функция `load()` явно проверяет, не достигнут ли конец файла. Делает она это с помощью вызова функции `feof()`. Это приходится делать потому, что `fread()` и в случае ошибки, и при достижении конца файла возвращает одно и то же значение.

Далее показана вся программа, обрабатывающая списки рассылки. Ее можно использовать как ядро для дальнейших расширений, в нее, например, можно добавить средства поиска адресов.

```
/* Простая программа обработки списка рассылки,
   в которой используется массив структур. */
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

struct addr {
  char name[30];
  char street[40];
  char city[20];
  char state[3];
  unsigned long int zip;
} addr_list[MAX];

void init_list(void), enter(void);
void delete(void), list(void);
void load(void), save(void);
int menu_select(void), find_free(void);

int main(void)
{
  char choice;

  init_list(); /* инициализация массива структур */
  for(;;) {
    choice = menu_select();
    switch(choice) {
      case 1: enter();
        break;
      case 2: delete();
        break;
      case 3: list();
        break;
      case 4: save();
        break;
      case 5: load();
        break;
      case 6: exit(0);
    }
  }

  return 0;
}

/* Инициализация списка. */
void init_list(void)
{
  register int t;

  for(t=0; t<MAX; ++t) addr_list[t].name[0] = '\0';
}

/* Получения значения, выбранного  в меню. */
int menu_select(void)
{
```

```
  char s[80];
  int c;

  printf("1. Ввести имя\n");
  printf("2. Удалить имя\n");
  printf("3. Вывести список\n");
  printf("4. Сохранить файл\n");
  printf("5. Загрузить файл\n");
  printf("6. Выход\n");
  do {
    printf("\nВведите номер нужного пункта: ");
    gets(s);
    c = atoi(s);
  } while(c<0 || c>6);
  return c;
}

/* Добавление адреса в список. */
void enter(void)
{
  int slot;
  char s[80];

  slot = find_free();

  if(slot==-1) {
    printf("\nСписок заполнен");
    return;
  }

  printf("Введите имя: ");
  gets(addr_list[slot].name);

  printf("Введите улицу: ");
  gets(addr_list[slot].street);

  printf("Введите город: ");
  gets(addr_list[slot].city);

  printf("Введите штат: ");
  gets(addr_list[slot].state);

  printf("Введите почтовый индекс: ");
  gets(s);
  addr_list[slot].zip = strtoul(s, '\0', 10);
}

/* Поиск свободной структуры. */
int find_free(void)
{
  register int t;

  for(t=0; addr_list[t].name[0] && t<MAX; ++t) ;

  if(t==MAX) return -1; /* свободных структур нет */
  return t;
}

/* Удаление адреса. */
void delete(void)
{
  register int slot;
  char s[80];

  printf("Введите № записи: ");
  gets(s);
  slot = atoi(s);

  if(slot>=0 && slot < MAX)
    addr_list[slot].name[0] = '\0';
}

/* Вывод списка на экран. */
void list(void)
{
  register int t;

  for(t=0; t<MAX; ++t) {
    if(addr_list[t].name[0]) {
```

```
      printf("%s\n", addr_list[t].name);
      printf("%s\n", addr_list[t].street);
      printf("%s\n", addr_list[t].city);
      printf("%s\n", addr_list[t].state);
      printf("%lu\n\n", addr_list[t].zip);
    }
  }
  printf("\n\n");
}

/* Сохранение списка. */
void save(void)
{
  FILE  *fp;
  register int i;

  if((fp=fopen("maillist", "wb"))==NULL) {
    printf("Ошибка при открытии файла.\n");
    return;
  }

  for(i=0; i<MAX; i++)
    if(*addr_list[i].name)
      if(fwrite(&addr_list[i],
         sizeof(struct addr), 1, fp)!=1)
          printf("Ошибка при записи файла.\n");

  fclose(fp);
}

/* Загрузить файл. */
void load(void)
{
  FILE  *fp;
  register int i;

  if((fp=fopen("maillist", "rb"))==NULL) {
    printf("Ошибка при открытии файла.\n");
    return;
  }

  init_list();
  for(i=0; i<MAX; i++)
    if(fread(&addr_list[i],
       sizeof(struct addr), 1, fp)!=1) {
         if(feof(fp)) break;
         printf("Ошибка при чтении файла.\n");
    }

  fclose(fp);
}
```

Содержание | <<< | >>>
[Главная] [Гостевая]
