Файлы

Лектор: Артамонов Юрий Николаевич

Университет "Дубна"
филиал Котельники

Содержание

Понятие файла

Хранение данных в переменных и массивах является временным, все эти данные теряются при завершении работы программы. Для постоянного хранения больших объемов данных используются файлы. Мы рассматривали понятие структуры в C как группы логически связанных полей, например, структура сотрудника организации может состоять из полей: фамилия, имя, отчество, должность, возраст и т.д. Файлом называется множество связанных структур (записей). Например, можно сохранить в файл все записи сотрудников какой-либо органинзации. Для облегчения поиска необходимых данных, по крайней мере одно поле в каждой записи файла выбирается в качестве *ключа записи* (например, в структуре сотрудника таким полем могло бы быть поле с фамилией, если нет однофамильцев). Существует множество способов организации записей в файле. Наиболее популярные из них - *последовательный файл* и *файл произвольного доступа*. В последовательных файлах записи хранятся в порядке определяемом ключом.

Понятие файла

С рассматривает любой файл как последовательный поток байтов. Каждый файл должен заканчиваться *маркером конца файла*, или особым байтом, определенным в самой программе. Когда файл открывается, ему ставится в соответствие некоторый *поток*. Как вы знаете, в начале выполнения программы автоматически открываются три потока и соответствующие им файлы: stdin, stdout, stderr. Потоки реализуют каналы передачи данных между файлами и программами. Открытый файл возвращает указатель на структуру FILE, которая определяется в заголовочном файле <stdio.h>. Эта структура включает в себя *дескриптор файла*, понимаемый как индекс из таблицы всех открытых файлов, за формирование которой отвечает операционная система. Например, stdin, stdout, stderr как раз являются дескрипторами на файлы стандартного ввода, вывода, вывода ошибок. В С есть многочисленные функции чтения данных из файла, или записи в файл. Например, функция fgetc, подобно getchar, считывает из файла один символ, соответственно fgetc(stdin) эквивалентно getchar. Аналогично fputc('a', stdout) эквивалентен putchar('a').

Создание файла последовательного доступа

В языке C программист сам должен заботиться о создании структуры файла, отвечающей требованиям конкретного приложения (что и в какой последовательности записывать в файл). Рассмотрим пример создания файла хранящего информацию о сотрудниках организации, который можно пополнять вновь прибывающими сотрудниками. Нас, как и раньше будет интересовать следующая информация: фамилия, имя, возраст, должность.

Пример создания файла последовательного доступа

```
#include<stdio.h>
main()
{
  int years;
  char first_name[30], last_name[30];
  char position[100];
  FILE *fpeople;
  if ((fpeople = fopen("test", "a")) == NULL)
    printf("Ошибка! Файл не может быть открыт для записи");
  else
    {
      printf("Введите фамилию, имя, возраст, должность сотрудника\n");
      scanf("%s%s%d%s",last_name,first_name,&years,position)
  ;
```

Пример создания файла последовательного доступа (продолжение)

```
      while (!feof(stdin))
  {
    fprintf(fpeople, "%s %s %d %s\n", last_name,first_name,
  years, position);
    printf("Введите фамилию, имя, возраст, должность сотрудника\n");
    scanf("%s%s%d%s",last_name,first_name,&years,position);
  }
      fclose(fpeople);
    }
  return 0;
}
```

Создание файла последовательного доступа - пояснение примера

Поясним работу данной программы.
1. В строке FILE *fpeople; объявляется указатель на структуру FILE. Программа на C управляет каждым файлом при помощи отдельной структуры FILE. Каждый открытый файл должен иметь отдельно объявленный указатель типа FILE, который используется для ссылок на файл.
2. Строка if ((fpeople = fopen("test "a")) == NULL) указывает имя файла «test», который будет использоваться программой, и устанавливает канал и режим общения с файлом. Это осуществляется с помощью функции fopen, которая имеет два параметра - имя файла и режим открытия файла. В нашем примере используется режим 'a' - добавление (если файл существует, новая информация добавляется к старой, если не существует, то файл создается). Также доступны режимы 'r' - только чтение, 'w' - запись с пересозданием заново (если файл с таким именем уже существует, он удаляется и создается вновь, вся старая информация утрачивается без предупреждения).

Создание файла последовательного доступа - пояснение примера

3. Строка while (!feof(stdin)) используется, чтобы опрерделить, установлен ли индикатор конца файла, на который ссылается stdin. Индикатор конца файла сообщает, что чтение данных закончено. В программе данный индикатор устанавливается, когда пользователь вводит соответствующую комбинацию клавиш. Функция feof возвращает ненулевое значение (true), если индикатор конца файла установлен (в Unix-подобных ОС комбинация конца файла - это Ctrl+D).
4. Оператор

```
fprintf(fpeople, "%s %s %d %s\n", last_name, first_name,
  years, position);
```

записывает данные в файл test, которые в дальнейшем могут быть считаны в программу обратно. Функция fprintf - это аналог printf, единственная особенность, fprintf требует первым аргументом использовать указатель на файл.

Создание файла последовательного доступа - пояснение примера

5. Чтобы корректно закончить работу с файлом его нужно закрыть, что осуществляется с помощью команды fclose(fpeople);, имеющей аргументом указатель на файл, который необходимо закрыть. Если функция fclose не вызывается явно, операционная система обычно сама закрывает открытые файлы при завершении работы программы. Однако настоятельно рекомендуется закрывать файл явно, как только выяснится, что программа больше не будет с ним работать.
Кроме уже перечисленных режимов 'r, w, a', возможны режимы:

- 'r+' открывает уже существующий файл для чтения, записи;
- 'w+' создает файл для чтения и записи (если файл уже существует, файл открывается и текущее содержание файла уничтожается;
- 'a+' открывает файл для чтения и записи, все записи производятся в конец файла. Если файл не сущесвует, он создается.

Чтение данных из файла последовательного доступа

Очевидно, что файлы создаются для того, чтобы при необходимости можно было бы извлечь и обработать хранящуюся в них информацию.
В целом режим чтения данных из файла последовательного доступа аналигичен уже рассмотренному режиму создания файлов с некоторыми отличиями. Ниже представлен соответствующий код.

```
#include<stdio.h>
main()
{
  int years;
  char first_name[30], last_name[30];
  char position[100];
  FILE *fpeople;
  if ((fpeople = fopen("test", "r")) == NULL)
    printf("Ошибка! Файл не может быть открыт для чтения");
```

Чтение данных из файла последовательного доступа

```
  else
    {
      fscanf(fpeople, "%s%s%d%s",last_name,first_name,&years
  ,position);
      while (!feof(fpeople))
  {
    printf("Фамилия: %s Имя: %s Возраст: %d Должность: %s\n",
  last_name,first_name, years, position);
    fscanf(fpeople, "%s%s%d%s",last_name,first_name,&years,
  position);
  }
      fclose(fpeople);
    }
  return 0;
}
```

Чтение данных из файла последовательного доступа - пояснение примера

```
if ((fpeople = fopen("test", "r")) == NULL)
```

Как видно, файл открывается для чтения.

```
fscanf(fpeople, "%s%s%d%s",last_name,first_name,&years,
  position);
```

Данный оператор считывает запись из файла. Функция fscanf - это аналог функции scanf, отличие лишь в том, что в fscanf требуется первый аргумент указатель на файл.
Для того, чтобы считать данные из файла последовательного доступа, программа обычно начинает чтение с начала файла и считывает все данные последовательно до тех пор, пока они не закончатся.

Чтение данных из файла последовательного доступа - пояснение примера

В ходе выполнения программы может возникнуть необходимость обработать данные из файла последовательного доступа несколько раз (начиная каждый раз с первой записи). Выполнение оператора

```
rewind(fpeople);
```

приводит к тому, что указатель позиции файла - показывающий номер байта, который должен быть считан следующим - будет перемещен на начало.
Отметим, что данные в файле последовательного доступа не могут быть изменены без риска разрушить другие хранящиеся в файле данные. Например, если вновь вводимая строка длиннее старой строки, то данные будут записаны поверх следующей записи файла, частично затерев их.
Поэтому последовательный доступ с помощью fprintf и fscanf обычно не используется для обновления записей в файле.Вместо этого переписывают весь файл целиком. Для этого копируют все данные и вновь перезаписывают уже с изменениями. Это очень неудобно при больших файлах.

Задание 9.1

Дано имя файла и целые положительные числа N и K. Создать текстовый файл с указанным именем и записать в него N строк, каждая из которых состоит из K символов «*» (звездочка).

Задание 9.2

Дан непустой текстовый файл. Удалить из него первую строку.

Задание 9.3

Создать файл, который необходимо заполнить случайными числами. Создать второй файл, в котором все числа из первого файла записаны в обратном порядке.

Задание 9.4

Создать базу простых чисел от 1 до 10000000, сохранив их в файл. Реализовать для заданного числа из диапазона от 1 до 10000000 проверку его на простоту.

Задание 9.5

Создать базу дружественных чисел от 1 до 1000000, сохранив их в файл ▸Ссылка из википедии о дружественных числах

Понятие файла произвольного доступа

Как было отмечено, в файле с последовательным доступом записи не обязаны иметь одинаковую длину. Напротив, отдельные записи в файле с *произвольным доступом* обычно имеют фиксированную длину, что позволяет получить к ним доступ непосредственно по индексу, без поиска по всем записям. В файл с произвольным доступом можно вставлять новые данные, не разрушая при этом те, что уже находятся в файле. Ранее сохраненные данные можно также обновить или удалить без переписывания всего файла.

Создание файла произвольного доступа

В отличие от обычной печати с помощью fprintf в файл последовательного доступа, в файлах произвольного доступа используется функция fwrite, которая пересылает в файл заданное число байт, начиная с указанного адреса памяти. Данные записываются с того места в файле, которое обозначено указателем позиции файла.
Формат использования данной функции следующий:
fwrite(&var,sizeof(type_var),count,pointer_file);
где var - переменная, значение которой мы хотим сохранить в файл;
type_var - тип переменной var;
count - в случае переменной count равен 1, если var - это массив, то count - это число элементов массива, которые следует записать на диск;
pointer_file - указатель на файл.
Для чтения из файла произвольного доступа используется функция fread, которая пересылает заданное число байт из места в файле, определенного указателем позиции файла, в массив памяти, начинающийся с указанного адреса.

Создание файла произвольного доступа (продолжение)

Функция fwrite записывает данные в формате с фиксированным размером, для этого они работают с «сырыми данными» (то есть в виде байтов), а не в привычном для человека формате, с которым оперируют printf, scanf.
Программы обработки файлов редко записывают в файл единственное поле. Обычно они записывают за один раз переменную типа struct.
Рассмотрим ряд примеров записи данных в файл произвольного доступа:
1. Сгенерировать N (значение N ввести с клавиатуры) вещественных чисел и записать их в файл.
2. Создать программу записи информации о клиенте, сохранив его фамилию, имя, дату рождения.

Создание файла произвольного доступа (пример 1)

```
#include<stdio.h>
#include <stdlib.h>
main()
{
   int N;
   printf("N=");
   scanf("%d", &N);
   srand(time(NULL));
   int i;
   float r;
   FILE *filedata;
   if ((filedata = fopen("data.dat", "w")) == NULL)
      printf("Не могу открыть файл для записи");
```

Создание файла произвольного доступа (пример 1)

```
  else
    {
      for (i=0;i<N;i++)
  {
    printf("%f\n",r = (float)(rand() % 100)/100);
    fwrite(&r, sizeof(float), 1, filedata);
  }
      fclose(filedata);
    }
  return 0;
}
```

Создание файла произвольного доступа (пример 2)

```
#include<stdio.h>
#include <stdlib.h>
struct client
{
  char last_name[15];
  char first_name[10];
  char birhday[10];
};

main()
{
  struct client clientdata = {"","",""};
  FILE *filedata;
  if ((filedata = fopen("data.dat", "w")) == NULL)
    printf("Не могу открыть файл для записи");
```

Создание файла произвольного доступа (пример 2)

```
  else
    {
      printf("Введите информацию по пяти клиентам\n");
      int i;
      for (i=1; i<=5; i++)
  {
    printf("Фамилия %d - го клиента: ", i);
    scanf("%s",&clientdata.last_name);
    printf("Имя %d - го клиента: ", i);
    scanf("%s",&clientdata.first_name);
    printf("Дата рождения %d - го клиента: ", i);
    scanf("%s",&clientdata.birhday);
    fwrite(&clientdata, sizeof(struct client),1, filedata);
  }
      fclose(filedata);
    }
  return 0;
}
```

Произвольная запись данных в файл произвольного доступа

Допустим в последнем примере данные о каком-либо клиенте требуется изменить. Для этого требуется доступ к заданной записи внутри файла произвольного доступа, что обеспечивает функция fseek. Функция fseek устанавливает указатель позиции файла в заданное положение (сдвигая его на заданное количество байт), после чего fwrite записывает данные в нужное место. Формат функции fseek следующий:
fseek(pointer_file, (number-1)*sizeof(struct), SEEK_SET);
где pointer_file - указатель на файл;
(number-1)*sizeof(struct) - выражение, вычисляющее смещение - номер байта, с которого начинается запись (поскольку нумерация байт начинается с нуля из номера данных number вычитается 1);
SEEK_SET - символическая константа, определяемая в заголовочном файле stdio.h, которая показывает, что указатель позиции файла устанавливается относительно начала файла на величину смещения. Кроме SEEK_SET, можно использовать SEEK_CUR - указывает, что смещение происходит относительно текущего положения в файле; SEEK_END - указывает, что поиск начнется с конца файла. Рассмотрим соответствующий пример.

Пример изменения произвольной записи

```
#include<stdio.h>
#include <stdlib.h>
struct client
{
  char last_name[15];
  char first_name[10];
  char birhday[10];
};

main()
{
  struct client clientdata = {"","",""};
  FILE *filedata;
  if ((filedata = fopen("data.dat", "r+")) == NULL)
    printf("Не могу открыть файл для записи");
```

Пример изменения произвольной записи (продолжение)

```
else
    {
      int n;
      printf("Введите номер клиента: "); scanf("%d",&n);
      fseek(filedata , (n-1)*sizeof(struct client), SEEK_SET);
      printf("Фамилия %d - го клиента: ", n);
      scanf("%s",&clientdata.last_name);
      printf("Имя %d - го клиента: ", n);
      scanf("%s",&clientdata.first_name);
      printf("Дата рождения %d - го клиента: ", n);
      scanf("%s",&clientdata.birhday);
      fwrite(&clientdata , sizeof(struct client),1, filedata);
      fclose(filedata);
    }
  return 0;
}
```

Последовательное и произвольное чтение данных из файла произвольного доступа

Для чтения данных из файла произвольного доступа используется функция fread, которая считывает заданное число байт из файла в память. Например, оператор
fread(&clientdata, sizeof(struct client), 1, filedata);
считает число байтов, равное sizeof(struct client), из файла, на который ссылается указатель filedata, и сохранит их в структуре clientdata. Байты считываются из файла, начная с места, определенного указателем позиции файла. Функцию fread можно использовать и для чтения нескольких элементов массива с фиксированным размером с помощью указателя на массив, в котором будут храниться элементы, и указанием числа элементов, которые необходимо прочитать. Используя fseek, можно прочитать выборочные данные. Рассмотрим соответствующие примеры.

Пример последовательного чтения из файла произвольного доступа

```
#include<stdio.h>
#include <stdlib.h>
struct client
{
  char last_name[15];
  char first_name[10];
  char birhday[10];
};

main()
{
  struct client clientdata;
  FILE *filedata;
  if ((filedata = fopen("data.dat", "r")) == NULL)
    printf("Не могу открыть файл для записи");
```

Пример последовательного чтения из файла произвольного доступа (продолжение)

```
{
        int i;
        for (i=1; i<6; i++)
  {
    fread(&clientdata, sizeof(struct client), 1, filedata);
    printf("Фамилия %d - го клиента: %s\n", i, clientdata.last_name)
  ;
    printf("Имя  %d - го клиента: %s\n", i, clientdata.first_name);
    printf("Дата рождения %d - го клиента: %s\n", i, clientdata.
  birhday);
  }
      fclose(filedata);
    }
  return 0;
}
```

Пример чтения фиксированной записи из файла произвольного доступа

```
#include<stdio.h>
#include <stdlib.h>
struct client
{
  char last_name[15];
  char first_name[10];
  char birhday[10];
};

main()
{
  struct client clientdata;
  FILE *filedata;
  if ((filedata = fopen("data.dat", "r")) == NULL)
    printf("Не могу открыть файл для записи");
```

Пример чтения фиксированной записи из файла произвольного доступа (продолжение)

```
else
    {
      int n;
      printf("Введите номер клиента: ");
      scanf("%d", &n);
      fseek(filedata, (n-1)*sizeof(struct client), SEEK_SET);
      fread(&clientdata, sizeof(struct client), 1, filedata);
      printf("Фамилия %d - го клиента: %s\n", n, clientdata.
  last_name);
      printf("Имя %d - го клиента: %s\n", n, clientdata.first_name);
      printf("Дата рождения %d - го клиента: %s\n", n, clientdata.
  birhday);
      fclose(filedata);
    }
  return 0;
}
```

Задание 9.6

Создать базу данных студентов группы. Каждая запись должна хранить фамилию, имя, массив оценок за контрольные работы, текущий рейтинг. Предусмотреть создание базы, модификацию базы, просмотр данных по условиям (список студентов с рейтингом не ниже заданного).