Файлы в C++

Понятие файла

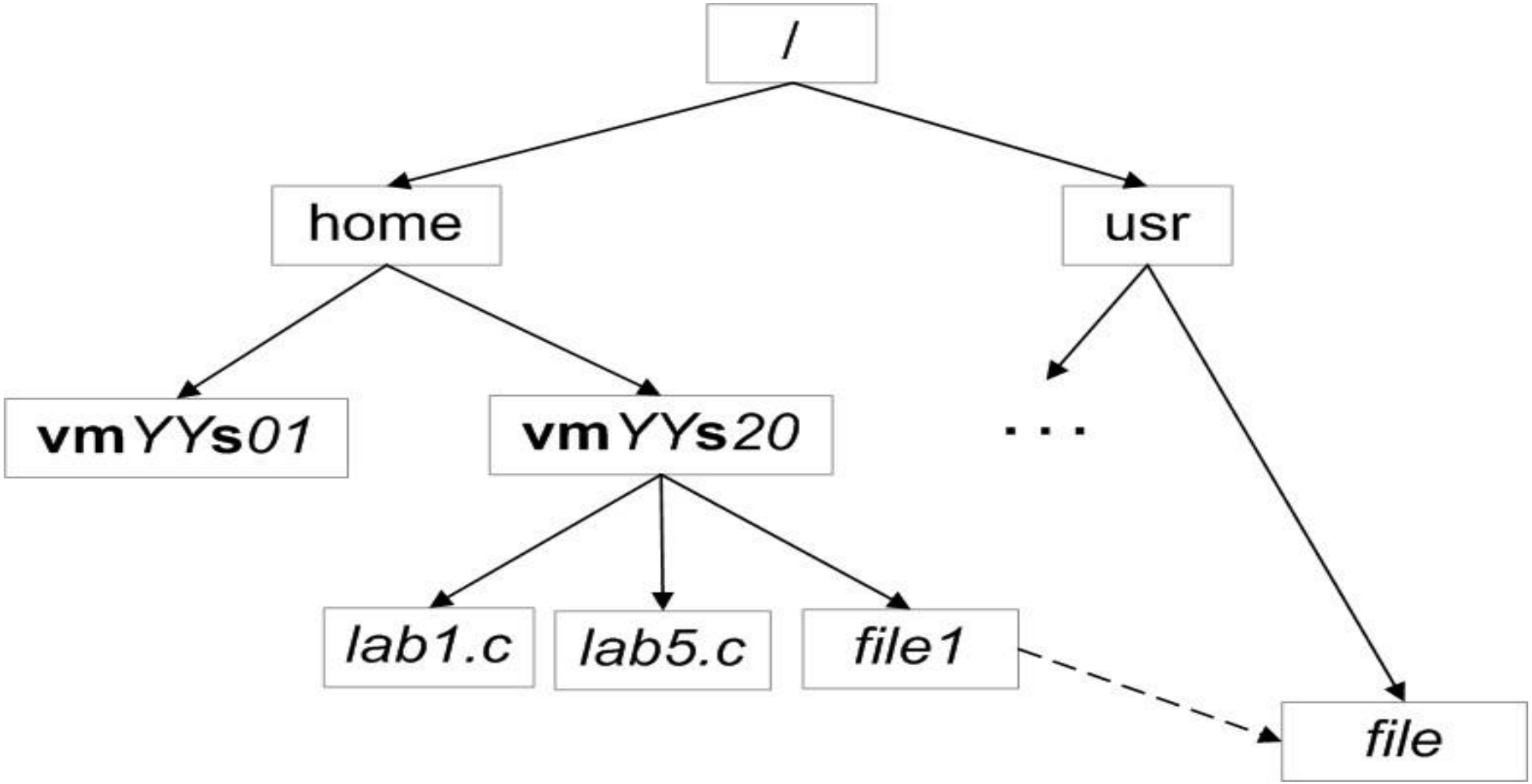
Файл (англ. file — папка, скоросшиватель) - сущность, позволяющая получить доступ к какому-либо ресурсу вычислительной системы и обладающая рядом признаков:

• фиксированное имя (последовательность символов, число или что-то иное, однозначно характеризующее файл);

• определённое логическое представление (формат) и соответствующие ему операции чтения/записи. Формат может быть любым — от последовательности бит до базы данных с произвольной организацией или любым промежуточным вариантом.

Файл (в информатике) — это упорядоченная совокупность данных, хранимая на диске и занимающая именованную область внешней памяти. Величина файла характеризуется объемом содержащимся в нем информации.

Файл и файловая система



Свойства файла

- *Имя файла (* В большинстве ФС имя файла используется для указания к какому именно файлу производится обращение)
- *Расширение имени файла (*Для определения типа (формата) файла)
- *Время (*Время создания; Время модификации; Время последнего доступа)
- *Владелец, группа и права доступа к файлу*

Операции с файлом: дополнительные сущности

- *хэндлер файла*, или *дескриптор* (описатель)
- *файловый указатель*
- *файловый буфер*
- *режим доступа*
- *режим общего доступа*

Операции, связанные с открытием файла

- *открытие файла,* обычно в качестве параметров передается имя файла, режим доступа и режим совместного доступа, а в качестве значения выступает файловый *хэндлер*, кроме того обычно имеется возможность в случае открытия на запись указать на то, должен ли размер файла изменяться на нулевой.
- *закрытие файла*, в качестве аргумента выступает значение, полученное при открытии файла. При закрытии все файловые буферы сбрасываются.
- *запись,* в файл помещаются данные.
- *чтение*, данные из файла помещаются в область памяти.
- *перемещение указателя* - указатель перемещается на указанное число байт вперёд/назад или перемещается по указанному смещению относительно начала/конца. Не все файлы позволяют выполнение этой операции (например, файл на ленточном накопителе может не «уметь» перематываться назад).
- *сброс буферов* - содержимое файловых буферов с не записанной в файл информацией записывается. Используется обычно для указания на завершение записи логического блока (для сохранения данных в файле на случай сбоя).

Операции, не связанные с открытием файла

- Удаление файла
- Переименование файла
- Копирование файла
- Перенос файла на другую файловую систему/носитель информации
- Создание симлинка или хардлинка
- Получение или изменение атрибутов файла

Файлы в C++

- Все файлы рассматриваются как неструктурированные последовательности байтов. Такой подход позволил распространить понятие файла и на различные устройства.
- Для программиста открытый файл представляется как последовательность считываемых или записываемых данных и с ним связывается поток ввода-вывода
- Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер).
- Содержимое файлов в C++ может быть представлено в текстовом и двоичном (бинарном) виде.

Общий алгоритм работы с файлами

- открытие потока;
- операции чтения/записи/смещения курсора…. обработка;
- закрытие потока

Работа с файлами в классическом стандарте C

Описать файловую переменную

Вся работа с файлом выполняется через *файловую переменную* - указатель на структуру типа FILE, определённую в стандартной библиотеке:

```
FILE *fp;
```

Открыть файл

FILE *fopen (char *имя_файла, char *режим_доступа)

Параметр **имя_файла** может содержать относительный или абсолютный путь к открываемому файлу:

1) "data.txt" - открывается файл data.txt из текущей папки

2) "f:\\my.dat" - открывается файл my.dat из головной папки диска f:

3) имя файла запрашивается у пользователя:

```
char buf[80];
printf ("\nвведите имя файла:");
fflush (stdin);
gets (buf);
```

Параметр **режим_доступа** определяет, какие действия будут разрешены с открываемым файлом (описаны в лабораторной работе №9)

Проверка открытия файла

Сравнить указатель, который вернула fopen, с константой NULL (nullptr) из стандартной библиотеки:

```
fp = fopen ("text.txt","r+b");
if (fp==NULL) {
 //Обработка ситуации "Не удалось открыть файл"
}
```

```
#include <windows.h>
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
 setlocale(LC_ALL,"Rus"); SetConsoleCP(1251); SetConsoleOutputCP(1251);
 FILE *fp;
 char buf[80];
 printf ("\nвведите имя файла:");
 fflush (stdin);
 gets (buf);
 fp = fopen (buf,"r+b");

 if (fp==NULL) {
  printf ("\n не удалось открыть файл");
  getchar();
  exit (1); //Выйти с кодом завершения 1
 }

 fflush(stdin); getchar();  return 0;
}
```

Когда в VS 2017 не работает fopen

1) заменить старые названия функций на их безопасные версии

```
FILE *out; fopen_s(&out,"data.txt", "wt");
```

2) в начало файла (до всех #include) включить директиву

```
#define _CRT_SECURE_NO_WARNINGS
```

Выбор способа чтения или записи данных: форматированный текст (красивый)

Обмен данными выполняют методами:

- fscanf - для чтения
- fprintf - для записи

Первым параметром этих функций указывается файловая переменная, в остальном работа совпадает со стандартными scanf и printf.

1 1.5 -3.5
2 3.5

Чтение текстового файла

```
FILE *fp = fopen ("text.txt","r");
if (fp==NULL) {
 printf ("\nне удалось открыть файл");  getchar(); exit (1);
}
float a;
while (1) {
 fscanf (fp,"%f",&a);
 if (feof(fp)) break; //Если файл кончился, выйти из цикла
 //здесь выполняется обработка очередного значения a,
```

например:

```
 printf ("%.2f ",a);
}
fclose(fp);
```

Чтение текстового файла: некоторые особенности

1. Очередное чтение данных изменяет внутренний *файловый указатель*. Этот указатель в любой момент времени, пока файл открыт, показывает на следующее значение, которое будет прочитано. Благодаря этому наш код с "бесконечным" while не зациклился.
2. Код показывает, как читать из файла заранее неизвестное количество значений – это позволяет сделать стандартная функция feof (проверка, достигнут ли конец файла; вернёт не 0, если прочитано всё).
3. Распространённый в примерах из Сети код вида

```
while (!feof(fp)) {
 fscanf (fp,"%f",&a);
 //обработка числа a
}
```

в ряде компиляторов может породить неточности при интерпретации данных. Например, этот код может прочитать как последнее значение завершающий перевод строки в файле, благодаря чему последнее прочитанное значение "удвоится".

Запись в текстовый файл

```
const int n=10;
int a[n],i;
FILE *fp=fopen ("result.txt","wt");
if (fp==NULL) {
 puts ("не удалось открыть файл на запись");
 getchar(); exit (1);
}
else {
 for (i=0; i<n; i++) a[i]=i+1;
 for (i=0; i<n; i++) {
  fprintf (fp,"%5d ",a[i]);
  if ((i+1)%5==0) fprintf (fp,"\n");
 }
 fclose (fp);
 //Закрыть файл, делать всегда, если в него писали
}
```

Важно! Ввод/вывод функциями библиотеки stdio.h буферизован, то есть, данные "пропускаются" через область памяти заданного размера, обмен данными происходит не отдельными байтами, а "порциями". Поэтому перед чтением данных желательно очищать буфер от возможных "остатков" предыдущего чтения, а после записи данных следует обязательно закрывать файл методом fclose, иначе данные могут быть потеряны.

Работа с неструктурированными текстовыми файлами (некрасивый текст)

Обмен данными выполняется с использованием функций:

fgetc и fputc - для посимвольного чтения и посимвольной записи данных;

fgets и fputs - для чтения и записи строк с указанным максимальным размером.

Как и в случае с функциями для чтения форматированных данных, у всех этих методов имеются аналоги для работы со стандартным вводом/выводом

Чтение неструктурированного текстового файла (в примере файл уже открыт)

Прочитать файл и определить длину каждой строки в символах

```
int c; int len=0,cnt=0;
while (1) {
 c=fgetc(fp);
 if (c=='\n') {
  printf ("\nString %d, length=%d",++cnt,len); len=0;
 }
 else len++;
 if (feof(fp)) break;
}
if (len) printf ("\nString %d, length=%d",++cnt,len);
```

Запись неструктурированного файла?

Работа с бинарными файлами

Основной способ работы с ним – чтение и запись наборов байт указанного размера.
Основные функции для чтения и записи бинарных данных – fread и fwrite соответственно. В базовой реализации они имеют по 4 параметра:

void *buffer - нетипизированный указатель на место хранения данных;
size_t (unsigned) size - размер элемента данных в байтах.
size_t count - максимальное количество элементов, которые требуется прочитать (записать);
FILE *stream - указатель на структуру FILE

Запись бинарного файла

```
FILE *fp=fopen ("data.dat","wb");
if (fp==NULL) {
 puts ("не удалось открыть файл");
 getchar(); exit (1);
}
const int n=10;
int a[n];
for(int i=0; i<n; i++) a[i]=i+1;
for (int i=0; i<10; i++) fwrite (&a[i],sizeof(int),1,fp);
 //Записали 10 эл-тов по одному
 //Если sizeof(int)=2, получим файл из 20 байт, если 4 - из 40
fclose (fp);
```

Чтение бинарного файла

```
unsigned char c;
//…
fread (&c,1,1,fp); //читаем по 1 байту
```

Функции прямого доступа в стандарте C (r+b)

- функции fgetpos и ftell позволяют выполнить чтение текущей позиции указателя в файле;
- функции fseek и fsetpos позволяют осуществить переход к нужной позиции в файле.

Функции прямого доступа (пример)

Определить размер файла в байтах, предположим, что файл уже открыт в режиме чтения или произвольного доступа.

```
seek (fp, 0, SEEK_END); //Встали на 0 байт от конца файла
long int pos;
pos = ftell (fp); //Получили текущую позицию в файле
if (pos<0) puts ("\nОшибка");
else if (!pos) puts ("\nФайл пуст");
else printf ("\nВ файле %ld байт",pos);
```

Работа с файлами в стандарте C ++

Открытие файлов

- для работы с файлами применяются классы ifstream для чтения, ofstream для записи и fstream для модификации файлов (добавление их осуществляется директивой #include)

- режим открытия файлов задает член данных перечисляемого типа open_mode, который определяется следующим образом:
  enum open_mode { app, binary, in, out, trunc, ate }; (их значение описано в лабораторной работе№9)

Примеры открытия файлов

```
ifstream file; file.open ("test.txt", ios::in |
ios::binary);

ofstream file;  file.open ("test.txt", ios::out |
ios::app);
```

Предполагается, что к проекту подключён соответствующий заголовочный файл:

```
#include <fstream.h>
```

Проверка при открытии: if (!file) { //Обработка ошибки открытия файла}

Операторы включения и извлечения

Оператор включения (<<) записывает данные в файловый поток

```
file << "Это строка текста";
file << "Это " << "строка " << "текста";
file << "Это строка текста" << endl;
```

С помощью оператора включения несложно записывать в файл значения переменных или элементов массива:

```
ofstream file ("Temp.txt");char  buff[] = "Текстовый массив содержит переменные";
int   vx = 100;
float pi = 3.14159;
file << buff << endl << vx << endl << pi << endl;
```

Temp.txt:

```
Текстовый массив содержит переменные
100
3.14159
```

Операторы включения и извлечения

Оператор извлечения (>>)производит обратные действия.

!!!!!

```
ifstream file ("Temp.txt");
char  buff[100];int   vx;float pi;
file >> buff >> vx >> pi;
```

Класс ifstream: чтение файлов

Метод	Описание
open	Открывает файл для чтения
get	Читает один или более символов из файла
getline	Читает символьную строку из текстового файла или данные из бинарного файла до определенного ограничителя
read	Считывает заданное число байт из файла в память
eof	Возвращает ненулевое значение (true), когда указатель потока достигает конца файла
peek	Выдает очередной символ потока, но не выбирает его
seekg	Перемещает указатель позиционирования файла в заданное положение
tellg	Возвращает текущее значение указателя позиционирования файла
close	Закрывает файл

Класс ofstream: запись файлов

Метод	Описание
open	Открывает файл для записи
put	Записывает одиночный символ в файл
write	Записывает заданное число байт из памяти в файл
seekp	Перемещает указатель позиционирования в указанное положение
tellp	Возвращает текущее значение указателя позиционирования файла
close	Закрывает файл

Добавление данных в текстовый файл с последующим чтением всего файла

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
ofstream file;
file.open("test.txt",ios::out|ios::app);
If (!file) {
cout << "File error - can't open to write data!";
cin.sync(); cin.get(); return 1; }
for (int i=0; i<10; i++) file << i << endl; file.close();
ifstream file2;
file2.open("test.txt", ios::in);
if (!file2) {
cout << "File error - can't open to read data!";
cin.sync(); cin.get(); return 2; }
int a,k=0;
while (1) {
file2 >> a;
if (file2.eof()) break;
cout << a << " ";  k++; }
cout << endl << "K=" << k << endl;
file2.close(); cin.sync(); cin.get();
return 0;}
```

Работа с бинарными файлами

Методы описаны выше, работа также как и с текстовыми

```
unsigned FileSize ( char *FileName)
{
ifstream File ( FileName, ios::in | ios::binary );
if ( !File ) // Проверили удалось ли открыть файл
{
cout << "Файл не найден! \n" ;
Return 0;
}
File.seekg (0, ios::end );
unsigned Size = File.tellg();
File.close ();
Return Size;
}
```