Тема 9
Организация ввода-вывода

Концепции организации ввода-вывода

В языке C++, как и в C, не предусмотрены какие-либо возможности для организации ввода-вывода. Весь ввод-вывод перенесен в специальные библиотеки.

Это позволило добиться эффекта ”независимости от платформы”, иначе говоря, получить аппаратно независимый язык разработки программ для различных платформ.

Два механизма ввода-вывода в C++

В программах на языке C++ можно равноправно использовать две библиотеки ввода-вывода:

- стандартную библиотеку функций ввода-вывода языка C (стандарт ANSI C);
- библиотеку потоковых классов, специально созданную для языка C++.

Эти библиотеки используют два принципиально различных механизма для организации ввода-вывода:

- первый основан на концепции файлов;
- второй работает со специальными классами, ориентированными на ввод-вывод – **потоковыми классами**.

Ввод-вывод в стиле ANSI C

Описания функций, а также определения переменных и констант задаются в файлах <stdio.h> или <cstdio>.

Некоторые переменные и константы:

- NULL = 0 – пустой указатель;
- EOF = -1 возвращается функциями ввода-вывода при достижении конца файла
- FILE – тип, описанный оператором struct. Поля этой структуры содержат информацию, необходимую для корректной работы с файлом. Программисту не требуется знать содержание этой структуры!

Файлы

Термин «файл» включает в себя два понятия:

- именованная совокупность данных, хранящаяся на внешнем носителе и обрабатываемая операционной системой, как единое целое;
- конструкция языка программирования, обеспечивающая работу механизмов ввода-вывода.

В рамках данной темы мы в основном будем рассматривать второе определение. Однако термин «**имя файла**» относится к первому определению!

Основные принципы работы с файлами

Файл – конечная совокупность данных на внешнем носителе. Данные из файла можно читать в память, а также можно записывать данные в файл из памяти. Перед этими операциями файл необходимо открыть, а после завершения – закрыть.

Чтение и запись файла производятся порциями.

Текущая позиция – смещение в байтах относительно начала файла той порции, которая была прочитана или записана.

Значение текущей позиции можно менять без операций чтения/записи.

Механизм чтения данных из файла

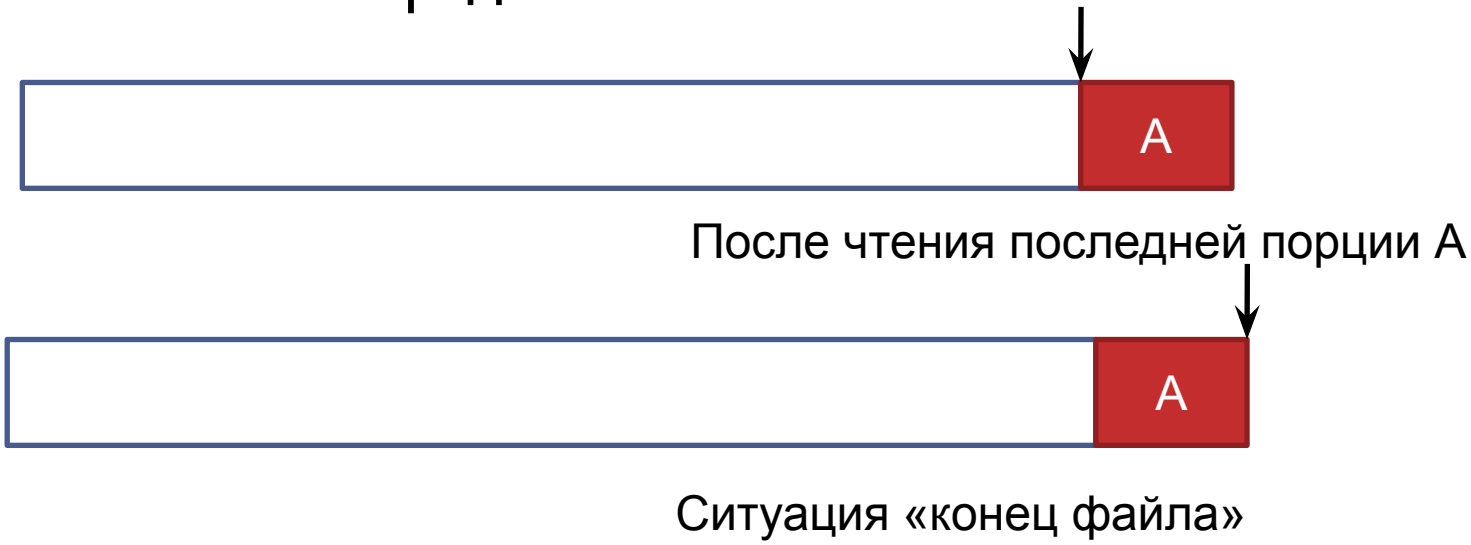
- Смещаем текущую позицию на начало читаемой порции;
- Читаем очередную порцию

После чтения порции A

После чтения порции B

Определение конца файла

- Прочитав последнюю порцию данных, мы ещё не знаем, что она последняя!
- Попытка следующего чтения вызывает ситуацию «конец файла». Это – ещё не ошибка, но результат чтения не определён!



Типы файлов

В языке C++ файлы делятся на 2 типа:

- текстовые
- бинарные

Текстовые файлы – совокупность строк (порция – строка). Строки отделяются друг от друга разделителями (в Windows – '\0x0D' и ' \0x0A '). Последняя строка может иметь разделители, а может и не иметь.

Для текстовых файлов возможен форматированный ввод/вывод.

Бинарные файлы - совокупность байт. Размер порции определяется в каждой операции чтения/записи. Форматированный ввод/вывод невозможен.

Описание файлов

Файлы описываются с помощью указателей на внутреннюю структуру FILE. Знание полей этой структуры не обязательно, поскольку работа с файлами обеспечивается на уровне соответствующих функций. Примеры описания файлов:

```
FILE *f1, *f2;
```

Программист имеет возможность работы с несколькими предопределенными файлами: stdin, stdout, stderr.

Открытие файлов

Функция открытия файла имеет формат

`FILE *fopen (const char* имяфайла, const char* режимоткрытия)`

В качестве режима открытия могут быть заданы следующие значения:

- r – существующий файл открывается для чтения;
- w – создается новый файл, который открывается для записи;
- a – существующий файл открывается для добавления информации в его конец;
- r+ – существующий файл открывается для чтения и записи;
- w+ – создается новый файл, который открывается для чтения и записи;
- a+ – существующий файл открывается для чтения и добавления информации в его конец.

Открытие файлов (продолжение)

Кроме того, в режиме открытия можно указывать символы t или b, которые соответствуют текстовому или бинарному файлу

Если открытие файла выполнено неудачно, функция fopen возвращает NULL.

Однако, если мы открыли пустой файл, мы ещё не знаем, что он пустой!

Для закрытия файла используется вызов функции fclose.

Канва программы, работающей с файлом

```
FILE *f;
if ((f = fopen(”myfile.txt”, ”rt”)) != NULL) {
   // работа с файлом
 fclose(f);
}
```

Другие функции для работы с файлами

Функция

```
int ferror(FILE*)
```

возвращает значение, не равное нулю, при возникновении ошибки ввода-вывода.

Функция

```
int feof(FILE*)
```

возвращает ненулевое значение, если достигнут конец файла.

Типы ввода/вывода

- ввод/вывод строк
- ввод/вывод символов
- форматированный ввод/вывод (только для текстовых файлов)

Ввод/вывод нуль-терминированных строк

Для чтения строки из файла и записи строки в файл используются функции

```
char *fgets(char *буфер, int длина, FILE *файл)
```

Функция возвращает NULL в случае ошибки или достижения конца файла

```
char *gets(char *буфер)
```

Чтение производится из файла stdin.

```
int fputs(char *буфер, FILE *файл)
```

Функция возвращает EOF в случае ошибки ввода-вывода

```
int puts(char *буфер)
```

Запись производится в файл stdout.

Пример работы с текстовыми файлами

Задача: текстовый файл input.txt содержит строки длиной не более 100 символов. Найти число строк, начинающихся с пробела.

```
int main() {
 int N = 0;
 FILE *f;
 char *S = new char[101];
 if ((f = fopen(”input.txt”, ”rt”)) != NULL) {
   while (fgets(S, 101, f)!= NULL)
    if (S[0] == ’ ’)
     N++;
   fclose(f);
   }
 cout << N;
 return 0;
}
```

Посимвольный ввод/вывод

При посимвольном вводе-выводе данные читаются из файла и записываются в файл **блоками**. За одну операцию чтения может быть прочитано несколько блоков (аналогичное правило действует для записи данных).

Как размер блока, так и их количество указывается при каждой операции ввода-вывода.

Функции посимвольного ввода/вывода

- Чтение данных:

```
size_t fread(void *буфер, size_t длинаблока,
             size_t числоблоков, FILE *файл)
```

- Запись данных:

```
size_t fwrite(const void *буфер, size_t длинаблока,
              size_t числоблоков, FILE *файл)
```

Обе функции возвращают число реально прочитанных или записанных блоков, так что результат их работы также можно использовать вместо проверки на конец файла

Целочисленный тип size_t используется для хранения информации о размере файла.

Длина блока и количество блоков

В функциях посимвольного ввода-вывода обычно указывается:

- Длина блока – 1, количество блоков – произвольное;
- Длина блока – произвольная, количество блоков – 1.

- Первый случай используется для того, чтобы мы могли правильно обработать последнюю (возможно, неполную) порцию данных
- Второй случай удобен, когда размер данных известен заранее (например, файл состоит из записей одинакового размера)

Пример работы с файлом, содержащим совокупность записей

Задача: Файл содержит несколько записей о студентах. Каждая запись имеет следующую структуру:

```
struct Student {
  char FIO [36];
  int kurs;
  int grup;
};
```

Посчитать количество записей о первокурсниках, хранящихся в файле.

Пример работы с файлом, содержащим совокупность записей (продолжение)

```
int main () {
  setlocale(LC_ALL, ".1251");
  Student *st = new Student;
  int N=0;
  FILE *f;
  if ((f=fopen("students.dat", "rb"))!=NULL) {
   while (fread(st, sizeof(Student), 1, f)) {
       if (st->kurs==1)
          N++;
   }
   fclose(f);
   cout << "Число первокурсников: " << N << endl;
  }
  else
   cout << "Ошибка открытия файла!" << endl;
  delete st;
  return 0;
}
```

Форматированный ввод/вывод

При форматированном вводе/выводе происходит преобразование данных, так что в памяти и в файле данные хранятся в различных форматах .

Примеры преобразования:

- перевод чисел в другую систему счисления:
- преобразование строк

Функции форматированного ввода/вывода

```
int scanf(const char * формат, [ адреса_переменных ] …)
int fscanf(FILE* файл, const char * формат, [ адреса_переменных ] …)
int printf(const char * формат, [ выражения ] …)
int fprintf(FILE* файл, const char * формат,
   [ выражения ] …)
int sprintf(char * буфер, const char * формат,
   [ выражения ] …)
```

Эти функции перед выводом преобразуют заданную **строку формата** в соответствии с содержащимися в ней **спецификациями формата** либо извлекают из прочитанной строки данные в переменные с указанными адресами.

Примеры спецификаций формата

Спецификация	Описание	Образец вывода		
		данные	формат	результат
%d, %i	Вывод целых чисел со знаком в десятичной системе счисления	27	%d %+d %7d %-7d %07d	27 +27 ~~~~~27 27~~~~~ 0000027
%u	Вывод целых чисел без знака в десятичной системе счисления			
%s	Вывод строк	Привет	%s %-10s %10s %.3s %-10.3s %10.3s	Привет Привет~~~~ ~~~~Привет При При~~~~~~~ ~~~~~~~При
%f	Вывод чисел с плавающей точкой	3.14159265	%f %11f %11.8f %-11.4f	3.141593 ~~~3.141593 ~3.14159265 3.1416~~~~~

Примеры спецификаций формата (продолжение)

Спецификация	Описание	Образец вывода		
		данные	формат	результат
%x, %X	Вывод чисел в шестнадцатеричной системе счисления (строчными или прописными буквами)	`27`	`%x` `%X`	`1b` `1B`
%e	Вывод чисел с плавающей точкой в экспоненциальной форме	`3.14159265`	`%e` `%11.2e`	`3.142e+000` `~~3.14e+000`

Пример работы форматированного ввода/вывода

Задача: В заданной матрице чисел типа double определить максимальный элемент в каждой строке. Нумерация строк для пользователя начинается с единицы.

Строки матрицы заполняются случайными числами.

Пример работы форматированного ввода/вывода (продолжение)

```
int main () {
  setlocale(LC_ALL, ".1251");
  double Matr [10][10];
  double Max;
  srand((unsigned int) time(NULL));
  for (int i =0; i<10; i++)
   for (int j=0; j<10; j++)
      Matr[i][j] = (double) rand()/13;
  for (int i =0; i<10; i++) {
   Max = Matr[i][0];
   for (int j=1; j<10; j++)
      if (Max < Matr[i][j])
         Max = Matr[i][j];
printf("максимальный элемент в %d-й строке равен %f\n",
  i+1, Max);
  }
  return 0;
}
```

Прямой доступ к файлу

Для прямого доступа к содержимому файла используются функции

```
long ftell(FILE* файл)
int fseek(FILE* файл, long смещение, int начало)
```

Первая из этих функций возвращает **текущую позицию** файла (в байтах, счет начинается с нуля), т. е. ту позицию, с которой будет выполнена следующая операция чтения-записи.

Вторая функция перемещает указатель текущей позиции на указанное смещение.

Прямой доступ к файлу (продолжение)

Параметр «начало» может иметь значения следующих констант:

```
SEEK_CUR – относительно старой текущей позиции;
SEEK_SET – относительно начала файла;
SEEK_END – относительно конца файла.
```

Функция fseek возвращает 0 в случае успешного завершения, так что запись

```
if (!fseek(f1, 0, SEEK_END))
  filesize = ftell(f1);
```

позволяет получить информацию о длине файла в байтах.

Переназначение файлов

Для переназначения файлов можно использовать функцию

```
FILE *freopen(const char *имя_файла, const char *
  режим_открытия, FILE * старый_файл );
```

Пример:

```
freopen(”gratis.in”, ”rt”, stdin);
freopen(”gratis.out”, ”wt”, stdout);
```

- Тема «Работа с потоками» переносится на самостоятельное изучение!