IX

C++: какую часть изучили

На настоящий момент были разобраны следующие возможности C++ для написания программ[1]

- фундаментальные типы (**char**, **int**, **double** и другие);
- специальные типы: статические массивы, ссылки и указатели;
- управляющие конструкции: условный переход, **switch** и циклы;
- определение собственных функций: передача параметров и возращаемые значения;
- пользовательские типы: псевдонимы и составные типы (стркутуры[2], перечисления, объединения).

[1] не забываем ещё про пространства имён и препроцессорные директивы

[2] как составные объекты, включающие в себя набор значений других различных типов

C++: какую часть изучили

Всех перечисленных возможностей языка программирования достаточно для написания программ в *процедурном стиле*.
Под этим словосочетанием понимают подход, в котором для решения конкретной задачи выбирются подходящие типы (встроенные или пользовательские, в зависимости от возможностей языка) и для операций над значениями переменных используется набор функций (определяемые самостоятельно или взятые из библиотек).
В рамках такого стиля реализована вся текущая стандартная библиотека языка C, которая доступна и из C++.

Кроме того, перечисленные выше возможности это всё, что предоставляет **язык программирования C** (не C++, не путать) для написания программ. За исключением следующих различий с C++:

- отсутствие ссылочных типов: только указатели;
- только одна форма для инициализации переменных — использование оператора «=»:

```
1 int value = 10;
2
3 double reals[] = {1.0, 2.0, 3.0};
```

- работа с функциями: аргументы в функцию передаются **только по значению** (перадача статических массивов по значению происходит аналогично таковой в C++);

C++: различия с языком C. Часть II

- в языке C отсутствует перегрузка функций — каждая функция должна иметь уникальный идентификатор. Другими словами, на сигнатуру функции не влияет список её параметров. Для примера,

```
1 int balance = -5;
2 printf("|%d| = %d\n", balance, abs(balance));
3
4 double rate = -0.3;
5 printf("|%.2f| = %.2f\n", rate, fabs(rate));
```

  В **C** для вычисления модуля числа с плавающей точкой используется функция **fabs**, тогда как в C++ реализована перегруженная версия для **abs**;

- для того, чтобы определить функцию, не принимающую никаких значений, необходимо в списке параметров явно использовать ключевое слово **void**:

C++: различия с языком C. Часть III

```
1 int f1(void) { return 10; }
2 int f2() { return 5; }
3
4 printf("f1() = %d\n", f1());
5 //printf("f1() = %d\n", f1(2, 3));
6 printf("f2() = %d\n", f2());
7 printf("f2() = %d\n", f2(1, 'c', "str"));
```

Пятая строка вызовет ошибку компиляции в C, но седьмая — вполне корректна. В C функция с пустым списком параметров может принима вообще любые аргументы. C++ такого не позволяет.

- только один способ создания псевдонимов — ключевое слово **typedef**;
- **полностью отсутствуют** пространства имён;
- ключевое слово **struct** входит в полное название структурного типа:

```
1 struct Account
2 {
3   int balance;
4   char name[80];
5   char address[200];
6 };
7
8 struct Account item = {25, "Joe", "Green st, ↩
    10"};
```

Не всегда хочется в каждом месте набирать слово «**struct**», поэтому один из популярных подходов к определению структурного типа состоит в одновременном определении и псевдонима на создаваемый тип:

```
1 typedef struct Account
2 {
3   int balance;
4   char name[80];
5   char address[200];
6 } Account;
7
8 Account item = {25, "Joe", "Green st, 10"};
```

Формально, в C был создан структурный тип «**struct Account**» и для него объявлен псевдоним с названием «**Account**»;

- инициализация структур с использованием имён полей:

```
1 Account item = { .name = "agent Smith",
2                  .balance = 100,
3                  .address = "whole world"};
```

Как показывает пример, при такой инициализации не обязательно сохранять порядок полей. В C++ подобный синтаксис официально станет доступным только с выходом стандарта C++20 (в конце 2020 года), но будет ограничен: порядок следования полей не может быть нарушен в инициализаторе.

- для явного приведения типов используется единственная конструкция:

```
1 int ival = 5;
2 double real = (int) ival;
```

- для работы с динамической памятью в стандратной библиотеке C определены функции **malloc**, **calloc**, **realloc** и **free**;
- отсутствует ключевое слово **nullptr**. Для обозначения *нулевого указателя* используется *макрос* **NULL**.

C++: различия с языком C

Были перечислены основные синтаксические различия. С их учётом, разобравшись в ограниченной части языка C++, вы имеете возможность и изучать, и реализовывать программы на языке C. Два в одном, так сказать. В дополнении, тот же препроцессор работает в обоих языках одинакого.
Умопинание **C** обусловлено тем, что далее переходит к рассмотрению основ работы системы ввода-вывода в C++. И практическая часть по работе с вводом-выводом будет разобрана с использованием средств из библиотеки <**cstdio**>, оставаясь в рамках процедурного подхода.

Основы ввода - вывода в C++

Ввод/вывод: что происходит

В далёкие до-дистанционные времена как-то раз разговор заходил о выделении общности у различных программ: они все получают некоторые входные данные, применяют набор вычислений, выдают в качестве результата набор выходных данных. В рамках лабораторных и источником входных данных, и приёмником выходных — выступала консоль (командная строка в терминах ОС Windows). Очевидно, это не является единственным способом: программы с графическим интерфейсом на различных устройствах, работа с файлами, работа с сетью представляют собой другие способы получения/отправки некоторых данных. И в языках программирования при проектировании стандартной библиотеки ставится вопрос: можно ли каким-нибудь образом обобщить работу с различными источниками/приёмниками данных?

Ввод/вывод: что происходит

?

Данные в программе:
* Текст: **char**
* Числа: **int, double, size_t**

Данные снаружи:
* Байты
  -> текст (ASCII, UTF8, ...)
  -> бинарные данные (raw bytes)

Ввод/вывод: что происходит

Есть условная программа на C++, есть источник/приёмник данных. В рамках программы все значения типизированы, вне её — фактически все данные являются набором байт, которые необходимо получить, преобразовать и сохранить в переменные.
Для получения-преобразования C++, как и многие другие языки программирования, использует концепцию **потоков данных**.

Ввод/вывод: кто работает

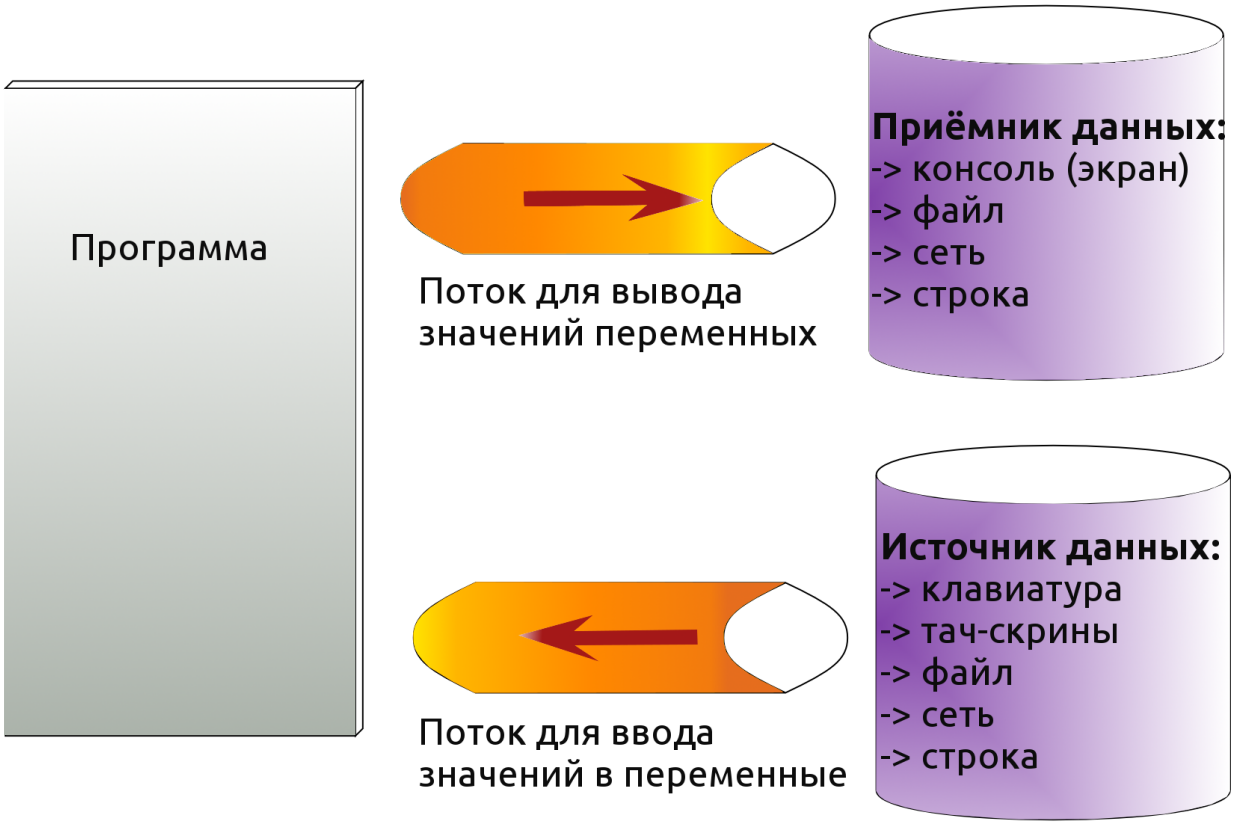
Идея в следующем:

Концепция **потока данных (stream)**

- Логическая сущность, связывающаяся с одним приёмником и/или источником данных
- Берёт на себя обязанности по приёму/отправке байт
- Преобразует байты в значения требуемых типов при вводе; преобразует значения различных типов в байтовое представление при выводе
- Скрывает взаимодействие с реальным устройством ввода-вывода

Ничего не говорим о конкретной реализации. Пока поток выступает на некоторая абстракция.
Соответственно, общая диаграмма становится яснее.

Потоковый ввод/вывод



Данные в программе:
* Текст: **char, char[]**
* Числа: **int, double, size_t**

Данные снаружи:
* Байты
  -> текст (ASCII, UTF8, ...)
  -> бинарные данные (raw bytes)

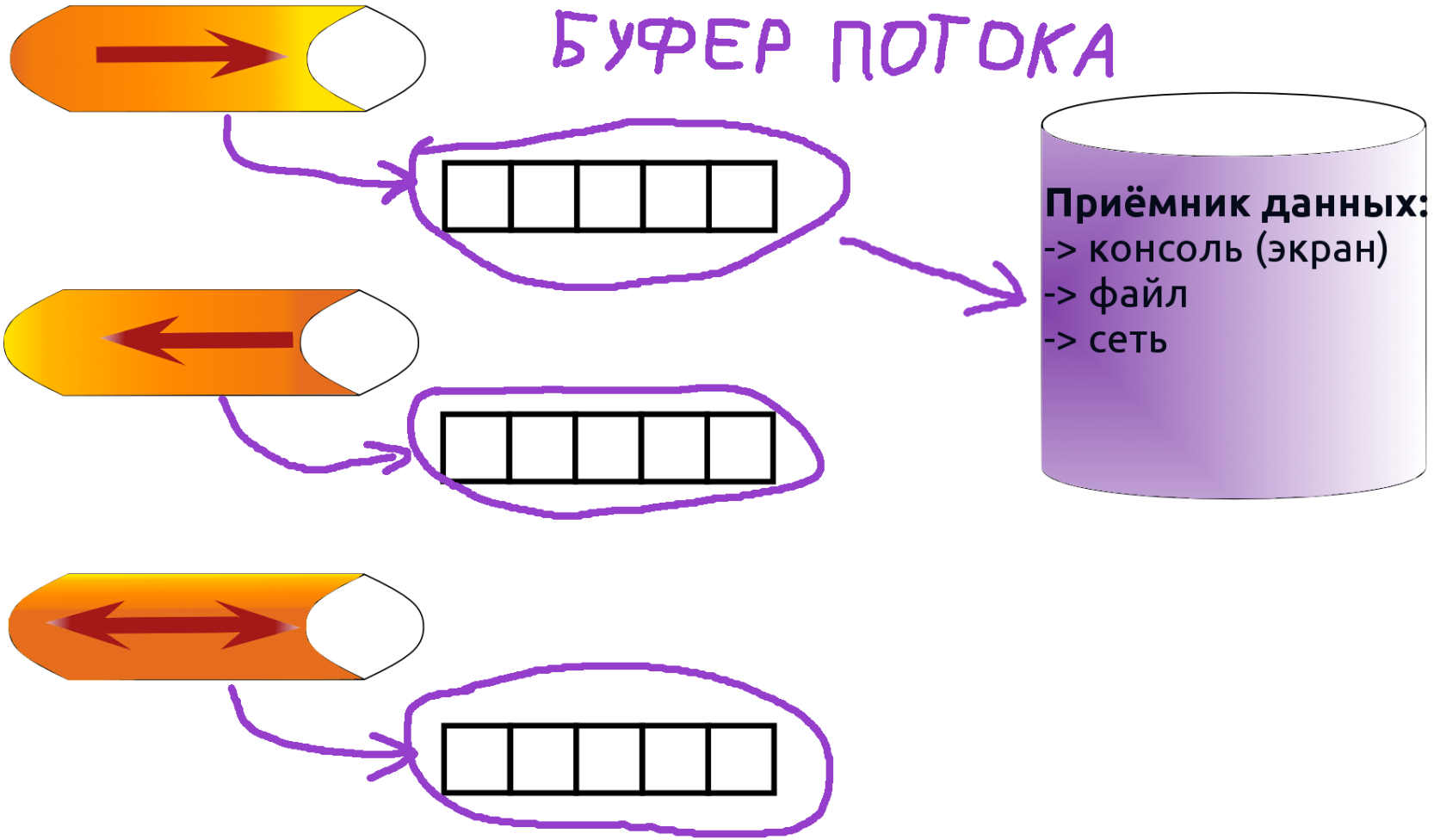
Потоковый ввод/вывод

По направлению передачи информации, можно выделить три вида потоков:

1. **поток вывода** используется только для вывода информации из программы во внешнее устройство. На предыдущем слайде стрелка от программы к приёмнику данных;
2. **поток ввода** — соответственно только ввод данных откуда-нибудь. Противоположная стрелка;
3. **двухсторонний поток** или поток ввода-вывода: позволяет одновременно и загружать данные из некоторого условного хранилища, и записывать данные в него. На предыдущем слайде не представлен.

Кроме того, возвращаясь к C++, потоки по умолчанию являются **буферизованными**.

Сначало схема



Теперь пояснения.
Под **буфером** потока понимается некоторый промежуточный массив байт, куда данные будут записаны перед тем, как произойдёт нужная операция с ними (в случае потока ввода под операцией понимается разбор данных программой; в случае потока вывода — реальная запись данных в приёмник).
В общем случае, можно считать, что буфер служит для оптимизации низкоуровневых операций ввода-вывода в ОС.
На примере записи в файл, гораздо оптимальнее накопить некоторый массив байт, скажем размера 1024, и за одну операцию его записать на реальное устройство (жесткий диск, флеш-память, всякие микро SD), чем 1024 раз вызывать операцию записи для каждого байта.

Неявно с буфером ввода сталкивались и при запросе данных с консоли. Вот есть код:

```
1 int val1, val2;
2 printf("Enter two numbers: ");
3 scanf("%d %d", &val1, &val2);
```

Если при запуске программы сделать ввод таким:

```
Enter two numbers: 4 -5 101 8.8 102
```

То все символы, начиная с **101**, останутся в связанном с консольным вводом буфере после вызова функции **scanf**. Если больше запроса данных не происходит, буфер будет отчищен по завершении работы программы.
И здесь первый факт, связанный с потоками в C++: при запуске программы ей предоставляется **три** потока ввода/вывода, связанные с консолью.

Потоковый ввод/вывод

При использовании библиотеки <**cstdio**> эти потоки будут связаны со следующими идентификаторами:

stdout: printf, putc, puts

stdin: scanf, getc, fgets

stderr: perror

Потоковый ввод/вывод

Технически говоря, **stdout**, **stdin** и **stderr** это три макроса, которые в зависимости от реализации разворачиваются в соответствующие потоки.

- **stdout** — стандратный поток вывода, используется, как правило, для печати информации в консоли;
- **stdin** — стандратный поток ввода, используется, как правило, для печати информации в консоли;
- **stderr** — стандратный поток вывода, используется для сообщения об ощибочной работе программы. По умолчанию, этот поток выводит символы в консоль, аналогично **stdout**.

Выше оговорка «как правило» использовалась из-за того, что операционные системы позволяют при запуске программы перенаправлять стандартные потоки ввода/вывода. Например, сделать вывод в файл, вместо окна консоли. Кому интересно, терзайте поисковики запросами вида «перенаправление стандартных потоков ввода вывода в C++».

Потоковый ввод/вывод

На слайде 21 для каждого потока указаны функции стандартной библиотеки, которые эти потоки используют. Так, **printf** всегда неявно использует поток **stdout**.
С потоком **stderr** связана функция **perror**. Она позволяет в некоторых случаях напечатать произвольное сообщение (строкового типа) с дополнительной информацией, что же произошло. Для примера,

```
1 double real = sqrt(-5.5);
2 if (std::isnan(real)) {
3   perror("NAN");
4 } // NAN: Numerical argument out of domain
5
6 real = std::pow(10000000.5, 100000000);
7 if (std::isinf(real)) {
8   perror("Infinity");
9 } // Infinity: Numerical result out of range
```

Комментарии на 4 и 9 строках показывают возможный вывод при запуске.

Потоковый ввод/вывод: файлы

Переходим к рассмотрению потокового ввода/вывода в файлы. Функции для создания файловых потоков ввода-вывода в стиле языка C определены в файле <**cstdio**>, также как **printf** или **scanf**.
Тип, с помощью которого происходят все манипуляции с потоком, это структура **FILE**. Она содержит набор полей, с помощью которых происходят действия с файлами. Сами по себе поля не предназначены для прямого обращения (и, в общем случае, набор полей может различаться у разных компиляторов), операции с файлами происходят через определяемые стандратной библиотекой функции.
Также из макросов, объявленных внутри <**cstdio**>, будет полезен **EOF**. Это макрос, разворачивающийся в *целое отрицательное число*. Согласно своему названию (EOF => End Of File) служит индикатором того, что все байты файла были считаны. Однако, в некоторых случаях используется как индикатор того, что произошла ошибочная ситуация.

Потоковый ввод/вывод: файлы

Функция **fopen** - создание потока для работы с конкретным файлом

```
FILE* fopen(const char *file_name,
            const char *mode);
```

- 1-ая строка **file_name** - название файла (полный путь)
- 2-ая строка **mode** - режим работы потока, состоящий из **типа операции** (ввод, вывод, всё вместе) и **дополнительных модификаторов** (читать информацию как текст или двоичные данные)
- Возращаемое значение: указатель на объект структуры **FILE** (представляющий поток), если создание потока прошло успешно, и **нулевой указатель** - в противном случае

Потоковый ввод/вывод: файлы

Функция **fopen** - режимы открытия (1):

mode	Что означает
"r", "rb"	**read**: создание потока для ввода (чтения) данных. Файл **должен** существовать, чтение начинается с начала. Если файл не существует — ошибка открытия
"w", "wb"	**write**: создание потока для вывода (записи) данных. Если файл уже существует, то всё его содержимое **удаляется**. Если не существует - создаётся новый файл
"a", "ab"	**append**: создание потока для вывода данных в конец файла (дозапись). Если файл не существует - создаётся новый. Невозможно записать в произвольное место в файле

Функция **fopen** - режимы открытия (2):

mode	Что означает
"r+", "rb+"	**update**: создание потока для одновременного ввода/вывода. Файл **должен** существовать, иначе — ошибка открытия. Запись возможна в произвольное место файла
"w+", "wb+"	**update**: одновременный ввод/вывод. Создаётся новый файл, либо удаляется содержимое существующего
"a+", "ab+"	**update**: одновременный ввод/вывод, но операции вывода осуществляются **только** в конец файла

Потоковый ввод/вывод: файлы

Буква «**b**» в значениях параметра **mode** означает открытие файлового потока в **двоичном (бинарном)** режиме. Без его указания подразумевается, что файл открыт в *текстовом режиме*. Любой файл представляет собой набор строк. В C++ в качестве разделителя строк используется символ «**\n**». Но различные ОС могут испоьзовать другую комбинацию символов для переноса строк в текстовых файлах. Так, ОС Windows использует два символа «**\r \n**» для обозначения переноса строки. Для того, чтобы соответствовать различным представлением, потоки открытые в *текстовом режиме* преобразуют ОС-специфичную комбинацию в «**\n**» при чтении файла, и осуществляют обратное преобразование при выводе информации в файл.

В отличие от текстового, в *двоичном режиме* никаких текстовых замен, аналогичных рассмотренной, производится не будет при любых операциях с потоком ввода или вывода.

Потоковый ввод/вывод: файлы

В отличие от ОС Windows, альтернативные ОС (основанные на Linux, MacOS, производные от BSD) не делают различий между текстовым и двоичным режимом (в них, как и в C++, перенос строки всегда осуществляется с помощью символа «**\n**»).
Противоположная открытию потока, функция **fclose** — закрытие существующего файлового потока (прекращение связи с файлом)

```
int fclose(FILE *stream);
```

- Аргумент **stream** - указатель на поток, подлежащий закрытию. **Не пытайтесь** передать в функцию нулевой указатель.
- Возвращаемое значение: **нуль**, если закрытие прошло успешно; значение **EOF** - в противном случае

Прежде, чем переходить к примерам.

Замечание об именах файлов

В зависимости от ОС, в именах файлов можено использовать как абсолютные, так и относительные пути, включающие в себя набор директорий (каталогов). Однако, функция **fopen** может создать файл только в **уже существующих** директориях. Если хоть одна директория, входящая в имя файла, не существует — фукнция **fopen** вернёт *нулевой указатель*. Если имя файла указано без пути (не включает никаких директорий), файл создаётся в той же директории, в которой запускается программа.

Прежде, чем переходить к примерам.

Дополнительные характеристики потоков

Среди полей структуры **FILE** есть некоторые, отвечающие за «состояние» потока.

- В первую очередь, это индикатор текущего положения потока: количество прочитанных (ввод)/записанных (вывод) байт в текущий момент.
- Такой индикатор позволяет в случае необходимости перемещаться по потоку (например, записывать значения в середину файла, а не в конец).
- Во вторую, структура скрывает индикатор ошибки при операциях ввода-вывода на потоке. Если такой индикатор установлен, все вызовы последующих операций на потоке закончатся неуспешно.

Потоковый ввод/вывод: файлы

Создание/закрытие файловых потоков: один из вариантов общего шаблона для работы

```
1 FILE *f_in_data = fopen("my_data_file.txt", "r");
2 if (f_in_data != nullptr) {
3   // Работаем с данными из файла
4   fclose(f_in_data);
5 } else {
6   perror("Open error");
7 }
```

Попытались установить связь с конкретным файлом через вызов **fopen**, проверили на нулевой указатель, если всё хорошо — после работы с данными закрыли связь с файлом через **fclose**.

Потоковый ввод/вывод: файлы

Можно выделить следующие типы ввода/вывода:

Ввод/вывод данных

↙ Форматированный	↓ Неформатированный	↘ Прямой
Для чтения: группа символов преобразуется в значение требуемого типа. Для записи: значения конкретного типа преобразуется в текстовый вид.	посимвольный ввод или вывод (как отдельных символов, так и группы)	читается или записывается строго определённое количество байт

Форматированный ввод/вывод

Функция **fprintf** - форматирование и вывод значений всех фундаментальных типов и строк в заданный файл

```
int fprintf(FILE *stream,
            const char *format_str, ...);
```

- Аргумент **stream** - указатель на поток, в который осуществляется запись. Должен быть открыть в соответствующем режиме
- Аргумент **format_str** - форматная строка, содержащая произвольные символы и спецификаторы выводимых значений (**%**-последовательности)
- **...** - переменное количество аргументов, равное количеству указанных в **format_str** спецификаторов, и соответствующие значения для вывода
- Возращаемое значение: в случае успеха - количество записанных байт; иначе - некоторое отрицательное значение.

Пример **fprintf** (название не зря похоже на **printf**)

```
1  FILE *f_out = fopen("some_results.txt", "w");
2
3  if (f_out != nullptr) {
4    int i_num = 567;
5    double r_num = 14.8326372364277;
6    char str[] = "This is so simple";
7
8    fprintf(f_out, "Integer: %+08d\n", i_num);
9    fprintf(f_out, "  Float: %+.5f\n", r_num);
10   fprintf(f_out, "String: {{%s}}\n", str);
11   fprintf(f_out, "%d * %d = %d", 4, 5, 20);
12
13   fclose(f_out);
14 } else {
15   perror("Open error");
16 }
```

В файле "some_results.txt"окажется текст:

```
Integer: +0000567
  Float: +14.83264
String: {{This is so simple}}
4 * 5 = 20
```

За исключением первого параметра (самого потока, в который осуществляем вывод), функция **fprintf** форматирует значение с помощью того же синтаксиса, что и **printf**.

Форматированный ввод/вывод

Функция **fscanf** - форматированный ввод значений различных типов из файлового потока

```
int fscanf(FILE *stream,
           const char *format_str, ...);
```

- **stream** - указатель на поток, из которого осуществляется чтение
- **format_str** - форматная строка, содержащая произвольные символы и спецификаторы вводимых значений (**%**-последовательности)
- **...** - переменное количество аргументов, равное количеству указанных в **format_str** спецификаторов, и соответствующие адреса переменных для записи вводимых значений
- Возращаемое значение: в случае успеха - количество записанных значений, равное количеству дополнительных аргументов; иначе - либо константа **EOF**, либо число меньшее количества доп. аргументов

Форматированный ввод/вывод

Пример **fscanf**: дан файл my_data.txt

```
45    678.905
1.2387E-3
Text, just ordinary text
```

```
1  FILE *f_data = fopen("my_data.txt", "r");
2  if (f_data != nullptr) {
3    int num1;
4    double real1, real2;
5
6    fscanf(f_data, "%d %lf", &num1, &real1);
7    fscanf(f_data, "%le", &real2);
8
9    printf("Integer: %d\n", num1);
10   printf("Reals: %f, %f\n", real1, real2);
11
12   char word[50];
13   fscanf(f_data, "%50s", word);
14   fclose(f_data);
```

```
15
16    puts("First word from file text line:");
17    puts(word);
18  } else {
19    perror("Open error");
20  }
```

За исключением создания потокового объекта и проверок, пример показывает, что работа функции **fscanf** полностью аналогична **scanf**. Преобразование групп символов в значения идёт по тем же самым правилам.

Справка про форматированный ввод/вывод есть много где в интернете, в том числе и здесь: https://github.com/posgen/OmsuMaterials/wiki/Format-output-in-C

Неформатированный ввод/вывод

Как сказано выше, неформатированный ввод/вывод оперирует отдельными байтами. А в C++ среди фундаментальных типов только один имеет размер, равный одному байту, то он и используется. Этот тип — **char** и его вариации, например беззнаковый вариант **unsigned char**.

Функция **fputc** - запись одного символа в файл

```
int fputc(int character, FILE *stream);
```

- Аргумент **character** - символ для записи. Принимается целочисленный код символа, но внутри функции происходит явное преобразование к **unsigned char**. Из-за этого, ожидаемое поведение будет только в том случае, если используются символы, чьи целочисленные коды не превышают значения **255**.
- Аргумент **stream** - указатель на поток, в который происходит запись
- Возвращаемое значение: в случае успеха — код символа; иначе — значение макроса **EOF** (которое в данном случае используется как индикатор неудачной записи)

Пример **fputc** - запись цифр в файл

```
1  FILE *f_digits = fopen("all_digits.dat", "w");
2  if (f_digits != nullptr) {
3    for (char sym = '0'; sym <= '9'; sym++) {
4      fputc(sym, f_digits);
5    }
6
7    fclose(f_digits);
8  } else {
9    perror("Open error");
10 }
```

Функция **fputs** - запись строк в файл

```
int fputs(const char *str, FILE *stream);
```

- **str** - строка для записи. Функция записывает всю переданную строку (до символа окончания строки). В отличии от **puts**, *не добавляет* символа переноса '\n'
- **stream** - указатель на поток, в который происходит запись
- Возращаемое значение: в случае успеха - некоторое неотрицательное число; иначе - константа **EOF**

Как видно, список параметров функции не очень гармонирует с **fprintf**: там поток передавался первым аргументом, здесь — последним.

Пример **fputs**

```
1  FILE *f_log = fopen("logfile.dat", "w");
2  if (f_log != nullptr) {
3    fputs("simulation part 1: success\n", f_log);
4    fputs("simulation part 2: success\n", f_log);
5    fputs("simulation part 3: failure\n", f_log);
6
7    fclose(f_log);
8  } else {
9    perror("Open error");
10 }
```

Уверен, после запуска примера с предыдущего слайда, не будет неожиданностью, что в файле "logfile.dat"окажутся три строки:

```
simulation part 1: success
simulation part 2: success
simulation part 3: failure
```

Неформатированный ввод/вывод

Аналогично выводу в файлы, определены функции для посимвольного ввода содержимого файлов.
Функция **fgetc** - ввод одного символа из файла

```
int fgetc(FILE *stream);
```

- **stream** - указатель на поток, из которого осуществляется чтение
- Возращаемое значение: в случае успеха - код текущего символа в файле; иначе — значение **EOF**. Опять же, внутри потока символ будет представлен значением типа **char**. Тип **int** используется только потому, что стандарт не гарантирует того, что символьный тип всегда будет знаковым. А макрос **EOF** производит подстановку отрицательного целого числа

Неформатированный ввод/вывод

Пусть есть файл text.txt с содержимым:

```
File with text info #1 2# #3 #4 5
#...#
```

```
1  FILE *f_text = fopen("text.txt", "r");
2  if (f_text != nullptr) {
3    int symb, sharp_count = 0;
4
5    do {
6      symb = fgetc(f_text);
7      putc(symb);
8
9      if (symb == '#') { ++sharp_count; }
10   } while (symb != EOF);
11   fclose(f_text);
12   printf("Sharp sign count: %d\n", sharp_count);
13 } else {
14   perror("Open error");
15 }
```

Функция **fgets** - ввод строки из файла

```
char* fgets(char *str, int max_count, FILE *stream)
```

- **str** - указатель на массив типа **char**, в который записываются извлекаемые символы
- **max_count** - максимально возможное количество символов, для записи в **str** с учётом символа окончания строки. Символ переноса строки '\n' - также может быть записан в **str**
- **stream** - указатель на поток, из которого осуществляется чтение
- Возращаемое значение: в случае успеха - указатель на **str**; иначе - **нулевой указатель**

Данная функция уже рассматривалась при разборе работы с текстом.

Пример **fgets**: дан файл vip_text.txt с текстом:

```
Morbi vitae nibh sed nisl bibendum imperdiet.

Morbi tincidunt ut lorem quis mollis. In
ligula nisl, sollicitudin et rutrum sed,
consectetur vitae libero.

Donec porta enim sem, vel interdum nisl
dapibus vitae. Vestibulum tempus scelerisque
blandit. Nunc posuere odio urna, eu ornare
urna ornare id. Phasellus imperdiet velit
viverra justo sodales efficitur. Donec a
aliquam urna, quis suscipit quam.
```

Так можно получить информацию порциями по 30 символов:

```
1  FILE *f_text = fopen("vip_text.txt", "r");
2  if (f_text != nullptr) {
3    char buf[30];
4
5    while (fgets(buf, 30, f_text) != NULL) {
6      printf("<<%s>>\n", buf);
7    }
8
9    fclose(f_text);
10 } else {
11   perror("Error again");
12 }
```

Ввод/вывод: сотояние потока

Функции, для проверки состояния потока ввода-вывода

- Достигнут ли конец файла?

```
int feof(FILE *stream);
```

- Произошли ли какие-либо ошибки при операциях ввода-вывода?

```
int ferror(FILE *stream);
```

Обе функции возращают **нуль**, если ответ на соответствующий вопрос отрицательный, и **не нулевое целое число** - если утвердительный.

Потоковый ввод/вывод: файлы

Задача: есть текстовый файл «source.dat» со случайными действительными числами. Нужно посчитать статистику, например - среднее и медиану

```
3.43434 53.5 43.546 4 6.776 0.86
1.546E2 8.986E-5 5.55 3.01
2.32 6.777
```

Для начала - выведем все числа из файла на экран.
На словах алгоритм действий таков: загружаем последовательно по одному действительному числу и печатаем с помощью **printf**

С учётом знаний об **feof** - реализация в лоб выглядит как

```
1  FILE *f_stat = fopen("source.dat", "r");
2
3  if (f_stat != nullptr) {
4    double cur_num;
5
6    while ( !feof(f_stat) ) {
7      fscanf(f_stat, "%lf", &cur_num);
8      printf("Number: %f\n", cur_num);
9    }
10 }
```

А вывод будет таков:

```
Number: 3.434340
Number: 53.500000
Number: 43.546000
...
Number: 2.320000
Number: 6.777000
Number: 6.777000
```

Притом, что в исходном файле число **6.777** представлено в единственном экземпляре. Так происходит, потому что после чтения последнего числа сам поток ещё «не знает», закончились ли символы в файле. Индикатор **EOF** будет установлен только при последующей попытке чтения из файла.

Потоковый ввод/вывод: файлы

Для того, чтобы чтение прошло так, как задумывалось, нужно быть уверенным в том, что **fscanf** завершилась без ошибок:

```
1  FILE *f_stat = fopen("source.dat", "r");
2
3  if (f_stat != NULL) {
4    double cur_num;
5
6    while ( !feof(f_stat) ) {
7      if ( fscanf(f_stat, "%lf", &cur_num) == 1 ) {
8        printf("Текущее число: %f\n", cur_num);
9      }
10   }
11 }
```

Будет введено ровно то количество чисел, что записаны в файле.

Исходная задача: среднее + медиана

```
1  size_t collect_nums(const char *f_name, double *&reals)
2  {
3    FILE *f_stat = fopen(f_name, "r");
4    if (f_stat == nullptr) { return 0; }
5    double cur_num; size_t cur_index = 0;
6    size_t nums_count = 0;
7
8    while ( !feof(f_stat) ) {
9      if (fscanf(f_stat, "%lf", &cur_num) == 1) {
10       nums_count++;
11       if (reals == nullptr) {
12         allocate(reals, nums_count) } else {
13         reallocate(reals, cur_index, nums_count);
14       }
15       reals[cur_index++] = cur_num;
16     }
17   }
18   fclose(f_stat);
19   return nums_count;
20 }
```

Потоковый ввод/вывод: файлы

```
21 struct Stat
22 {
23   double average;
24   double median;
25 };
26
27 Stat compute_stat(double *reals, size_t count)
28 {
29   double sum = 0.0;
30   for (size_t i = 0; i < count; i++) {
31     sum += reals[i];
32   }
33
34   std::sort(reals, reals + count);
35
36   return Stat{sum / count, reals[count / 2]};
37 }
```

Потоковый ввод/вывод: файлы

Первая функция читает числа из файла в динамический массив, вторая — подсчитывает статистику. Применение:

```
1 double *values = nullptr;
2 size_t count = collect_nums("source.dat" values);
3
4 if (count > 0) {
5   Stat result = compute_stat(values, count);
6   printf("average: %f\n", result.average);
7   printf("median: %f\n", result.median);
8 }
```

Для исходного файла с числами, вывод будет похож на:

```
average: 23.656119
median: 5.550000
```

Были использованы функции **allocate** / **reallocate** — это аналоги таковых из четвёртой лекции, только для указателей на **double**.

Потоковый ввод/вывод: файлы

Ещё пример: дан текстовый файл, запомнить все числа из него

```
В работе [60] проведено численное исследование
равновесной динамики неупорядоченной модели
Изинга с некоррелированными дефектами
для спиновых концентраций p = +0.8, +0.85 и
+0.65. При использовании конечномерного
скейлингового анализа для решеток с
12 < L < 64 получено значение критического
показателя z = 2.35(2) для системы с
концентрацией спинов p = 1 - 0.2. Спадание
корреяционной функции происходило с
показателем -1.578.
```

Потоковый ввод/вывод: файлы

Будут использованы ещё две функции из стандартной библиотеки:

- Вернуть уже прочитанный символ в поток:

```
int ungetc(int character, FILE *stream);
```

- Убрать индикатор конца файла и/или индикатор ошибки операций ввода/вывода:

```
void clearerr(FILE *stream );
```

- Проверить, является ли переданный символ - цифрой:

```
#include <cctype>

int isdigit(int character);
```

Потоковый ввод/вывод: файлы

Здесь уже не будет разбиение на функции, код представлен последовательно, как общая идея к решению подобной задачи.

```
1  FILE *source = fopen("text.txt", "r");
2
3  if (source == nullptr) {
4    perror("Open error");
5    return EXIT_FAILURE;
6  }
7
8  double *reals = nullptr;
9  size_t reals_cnt = 0, cur_index = 0;
10 char symb;
11
12 while ( (symb = fgetc(source)) != EOF ) {
13   if (isdigit(symb) || symb == '+'
14                     || symb == '-') {
15     ungetc(symb, source);
```

Потоковый ввод/вывод: файлы

Задача: дан текстовый файл, запомнить все числа из него

```
16      double tmp;
17      if (fscanf(source, "%lf", &tmp) == 1) {
18        reals_cnt = cur_index + 1;
19        if (reals == nullptr) {
20          allocate(reals, reals_cnt)
21        } else {
22          reallocate(reals, cur_index, reals_cnt);
23        }
24
25        reals[cur_index++] = tmp;
26      } else {
27        clearerr(source);
28        fgetc(source);
29      }
30    } // if (isdigit(...) ... )
31  } // while
```

Потоковый ввод/вывод: файлы

```
39 fclose(source);
40
41 if (reals != nullptr) {
42   printf("Найденные числа: ");
43   for (size_t i = 0; i < reals_cnt; ++i) {
44     printf("%.5f ", reals[i]);
45   }
46 }
```

И вывод:

```
Найденные числа: 60.000 0.800 0.850 0.650 12.000
64.000 2.350 2.000 1.000 0.200 -1.578
```

Прямой ввод/вывод

Прямой ввод/вывод - предназначен для записи/чтения строго определённого количества байт. Стандартная библиотека предоставляет следующие функции:

- Записать байты в файл

```
size_t fwrite(const void *ptr, size_t elem_sz,
              size_t elems_count, FILE *stream);
```

- Прочитать байты из файла:

```
size_t fread(void *ptr, size_t elem_sz,
             size_t elems_count, FILE *stream);
```

, где **ptr** - указатель на начало блока памяти (либо куда записываются байты, либо откуда берутся для вывода в файл); **elem_sz** - размер (в байтах) одного элемента для ввода/вывода, **elems_count** - общее количество элементов, **stream** - поток ввода/вывода.
Возращаемое значение: число, которое равно **elems_count** в случае успеха и не равно, иначе.

Прямой ввод/вывод

Задача: одной программой записать в файл заданное количество массивов целых чисел из 10 элементов. Второй программой - определить количество записанных массивов (сколько штук) и загрузить один из них по выбору

Прямой ввод/вывод

Запись 10-элементных массивов в файл. Здесь то двоичный режим и пригодится.

```
1  const size_t SZ = 10;
2  FILE *out_stream = fopen("arrays.bin", "wb");
3
4  if (out_stream != nullptr) {
5    int arr[SZ]; size_t how_many;
6    printf("Введите количество массивов: ");
7    scanf("%lu", &how_many);
8
9    for (size_t att = 1; att <= how_many; ++att) {
10     for (size_t i = 0; i < SZ; ++i) {
11       arr[i] = rand();
12     }
13     size_t recorded = fwrite(arr, sizeof(int), SZ, ↩
           out_stream);
14     if (recorded != SZ) { break; }
15   }
16   fclose(out_stream);
17 }
```

Прямой ввод/вывод

Чтение 10-элементных массивов из файла. Напомним, каждый экземпляр структуры **FILE** имеет поле, сохраняющее **позицию** в файле: на каком байте от начала файла находится связанный с ним поток (смещение происходит в результате операций ввода/вывода).

- Узнать текущую позицию потока

  ```
  long ftell(FILE *stream);
  ```

  В случае ошибки - функция вернёт значение **-1L**.

- Изменить позицию потока

  ```
  int fseek(FILE *stream, long offset, int from);
  ```

  offset - отступ от некоторой позиции в файле, заданной аргументом **from**. В качестве позиции используются три константы: **SEEK_SET** (начало файла), **SEEK_CUR** (текущая позиция), **SEEK_END** (конец файла).

Чтение 10-элементных массивов из файла.

Что нужно для второй программы?

1. Узнать количество массивов в файле
2. Запросить номер загружаемого массива
3. Считать нужный массив из файла

Чтение 10-элементных массивов из файла.

```
1  const size_t SZ = 10, ARR_BYTES = sizeof(int) * SZ;
2  FILE *in_stream = fopen("arrays.bin", "rb");
3
4  if (in_stream != nullptr) {
5    fseek(in_stream, 0, SEEK_END); // Шаг (1) начат
6    long how_many = ftell(in_stream) / ARR_BYTES;
7    if (how_many < 1) {
8      perror("No arrays in file"); exit(1);
9    }
10   fseek(in_stream, 0, SEEK_SET); // Шаг (1) выполнен
11
12   int arr_num = 0;  // Шаг (2) начат
13   do {
14     printf("Введите номер (всего - %ld): ", how_many);
15     scanf("%d", &arr_num);
16   } while (arr_num < 1 || arr_num > how_many);
17                     // Шаг (2) выполнен
```

```
18    arr_num--; // Для вычисления смещения. Шаг (3) начат
19    int arr[SZ];
20    fseek(in_stream, arr_num * ARR_BYTES, SEEK_SET);
21    size_t read = fread(arr, sizeof(int), SZ, in_stream);
22    fclose(in_stream); // Шаг (3) выполнен
23
24    if (read != SZ) {
25      perror("Elements count less than 10");
26      exit(1);
27    }
28
29    printf("Loaded array:\n  ");
30    for (size_t i = 0; i < SZ; ++i) {
31      printf("%d ", arr[i]);
32    }
33  }
```

Потоковый ввод/вывод: файлы

Ещё примеры на **ftell**/**fseek** и неформатированный ввод/вывод

github.com/posgen/OmsuMaterials/blob/master/2course/Programming/examples/8_file_operations_example/rewrite_example.c

github.com/posgen/OmsuMaterials/blob/master/2course/Programming/examples/8_file_operations_example/save_and_get_structs.c

Правда, оба примера реализованы на языке C, но преобразование к C++ версии не должно стать преградой для понимания.

Потоковый ввод/вывод: файлы

Пара полезных функций:

- передвинуть позицию потока в начало и отчистить индикаторы конца файла/ошибок на потоке:

  ```
  void rewind(FILE *stream);
  ```

  Фактически, делает аналог следующих вызовов:

  ```
  fseek(stream, 0, SEEK_SET);
  clearerr(stream);
  ```

- сброс буфера для потоков вывода:

  ```
  int fflush(FILE *stream);
  ```

  Принудительно записывает символы из буфера в файл. Возвращает значение **нуль**, если перация прошла без ошибок, и **EOF** в противном случае.

И более полная справка по использованию потоков в стиле C в C++: **en.cppreference.com/w/cpp/io/c**