**17-18 ОАП Лк6 Ввод и вывод данных. Оператор присваивания.**

**Составной оператор**

Традиционный ввод-вывод

Для использования традиционного ввода-вывода в духе C, в программу необходимо включить заголовочный файл <cstdio>. (Разумеется, компилятор должен иметь доступ к соответствующей объектной библиотеке для правильной сборки исполняемого файла.)

Библиотека *stdio* предоставляет необходимый набор функций для ввода и вывода информации как в текстовом, так и в двоичном представлении. Следует отметить, что в отличие от классической C‑библиотеки, в современных библиотеках имеются более безопасные аналоги «классических» функций. Как правило, они имеют такое же имя, к которому добавлен суффикс *\_s*. Рекомендуется использовать именно эти, безопасные функции.

Самая Первая Программа с использованием библиотеки *stdio* выглядит так:

#include <cstdio>

int main()

{

printf("Hello, world!\n");

}

При запуске консольного приложения неявно открываются три потока: *stdin* — для ввода с клавиатуры, *stdout* — для буферизованного вывода на монитор и *stderr* — для небуферизованного вывода на монитор сообщений об ошибках. Эти три символа определены посредством <cstdio>.

В *stdio* для консольного ввода-вывода предусмотрена отдельная группа функций. Однако эти функции, как правило, являются обёртками для аналогичных функций файлового ввода-вывода, для которых аргумент типа *FILE* задан по умолчанию.

Самая Первая Программа с использование файлового вывода из библиотеки *stdio* выглядит так:

#include <cstdio>

int main()

{

fprintf(stdout, "Hello, world!\n");

}

Некоторые популярные функции из *stdio*:

FILE \*fopen(const char \*filename, const char \*mode) *// открытие файла*

int fclose(FILE \*stream) *// закрытие файла*

int printf(const char \*format, ...) *// фоматированный консольный вывод*

int fprintf(FILE \*stream, const char \*formatб, ...) *// форматированный ввод из файла*

int sprintf(char \*s, const char \*format, ...) *// форматированный вывод в буфер (строку)*

int scanf(const char \*format, ...) *// фоматированный консольный ввод*

int fscanf(FILE \*stream, const char \*format, ...) *// фоматированный ввод*

int sscanf (const char \*s, const char \*format, ...) *// фоматированный ввод из буфера (строки)*

int fgetc(FILE \*stream) *// читает символ из файла*

char \*fgets(char \*s, int n, FILE \*stream) *// читает строку из файла*

int fputc(int с, FILE \*stream) *// записывает символ в файл*

int fputs(const char \*s, FILE \*stream) *// записывает строку в файл*

int getchar(void) *// читает символ из stdin*

char \*gets(char \*s) *// читает строку из stdin*

int putchar(int с) *// записывает символ в stdout*

int puts(const char \*s) *// записывает строку в stdout*

int ungetc(int с, FILE \*stream) *// возвращает символ обратно в файл для последующего чтения*

Сущность *FILE* представляет собой структуру, в которой хранится вся информация для управления потоком ввода-вывода.

Файл открывается функцией *fopen()*, которой передаются два параметра. Первый параметр определяет имя файла. Второй — определяет режим открытия файла: чтение, запись, произвольный доступ и т.п., а также указание на то, как работать с данными: в текстовом или двоичном режиме. Подробности — см. в документации.

Пример использования *stdio*

#include <cstdio>

#include <errno.h>

const char \*filename = "testfile.txt";

int main()

{

FILE \*fin, \*fout;

int ecode;

*// открытие файла для записи в текстовом режиме,*

*// запись данных и закрытие файла.*

if((fout = fopen(filename, "w")) != NULL) {

for (int i = 0; i < 16; i++) {

if ((ecode = fprintf(fout, "%d\n", i\*i)) <= 0) {

fprintf(stderr, "Write error in file \"%s\", code %d\n", filename, ecode);

fclose(fout);

return 1;

}

}

fclose(fout);

}

else {

fprintf(stderr, "Output file open error \"%s\", code %d\n", filename, errno);

return 1;

}

*// открытие файла для чтения в текстовом режиме,*

*// чтение данных, форматированный вывод на консоль, закрытие файла.*

int data;

int counter = 0;

if((fin = fopen(filename, "r")) != NULL) {

while ((ecode = fscanf(fin, "%d", &data)) != EOF) {

printf("%8d", data);

if (++counter % 4 == 0) {

putchar('\n');

}

}

if ((ecode = ferror(fin)) != 0) {

fprintf(stderr, "Read error in file \"%s\", code %d\n", filename, ecode);

fclose(fin);

return 2;

}

fclose(fin);

}

else {

fprintf(stderr, "Input file open error \"%s\", code %d\n", filename, errno);

return 2;

}

return 0;

}

Следует отметить, что существует еще одна библиотека, ориентированная исключительно на консольный ввод-вывод — <conio.h>.

Ввод-вывод с помощью потоков STL

Для использования объектно-ориентированного консольного ввода-вывода с помощью потоков (stream) STL в программу необходимо включить заголовочный файл <iostream>, а для файлового ещё и <fstream>. (Разумеется, компилятор должен иметь доступ к соответствующей объектной библиотеке для правильной сборки исполняемого файла.)

Самая Первая Программа с использованием потоков STL выглядит так:

#include <iostream>

using namespace std;

int main() {

cout << "Hello, world!\n";

}

При запуске консольного приложения неявно открываются четыре потока: *сin* — для ввода с клавиатуры, *сout* — для буферизованного вывода на монитор, *сerr* — для небуферизованного вывода на монитор сообщений об ошибках и *clog* — буферизованный аналог *cerr*. Эти четыре символа определены посредством <iostream>.

Потоки *cin*, *cout* и *cerr* соответствуют потокам *stdin*, *stdout* и *stderr* соответственно.

Иерархия классов ввода-вывода STL достаточно сложна. Любители тонких ощущений могут найти её описание в литературе. Впрочем, остальных также не минует чаша сия, но только позже, когда потребуются знания чуть повыше того базового уровня, который описывается здесь.

Для ввода-вывода сначала необходимо создать поток — экземпляр соответствующего класса STL, а затем связать его с файлом. Для потока вывода используется класс *ofstream*, для потока ввода — *ifstream*, для потока ввода-вывода — *fstream*. В каждом из этих классов есть метод *open()*, который связывает поток с файлом. Проще говоря, открывает файл. Методу передаются два параметра: имя файла и режим открытия файла. Второй параметр представляет собой набор битовых флагов, определяющих режим открытия файла (чтение, запись и пр.) и способ работы с данными (текстовый или двоичный режим). Второй параметр опционален, т.е. имеет значение по умолчанию, соответствующее классу.

ifstream::open(const char \*filename, ios::openmode mode = ios::in);

ofstream::open(const char \*filename, ios::openmode mode = ios::out | ios::trunc);

fstream::open(const char \*filename, ios::openmode mode = ios::in | ios::out);

Вышеупомянутые классы имеют также конструкторы, позволяющие открыть файл сразу при создании потока. Параметры этих конструкторов полностью совпадают с параметрами метода *open()*.

При ошибке открытия файла (в контексте логического выражения) поток получает значение *false*.

Файл закрывается методом *close()*. Этот метод также вызывается при разрушении экземпляров классов потоков.

Операции чтения и записи в поток, связанный с файлом, осуществляются либо с помощью операторов << и >>, перегруженных для классов потоков ввода-вывода, либо с помощью любых других методов классов потоков ввода-вывода.

Некоторые наиболее употребляемые методы:

*// чтение данных:*

getline() *// читает строку из входного потока.*

get() *// читает символ из входного потока.*

ignore() *// пропускает указанное число элементов от текущей позиции чтения.*

read() *// читает указанное количество символов из входного потока и сохраняет их в буфере (неформатированный ввод).*

*// запись данных:*

flush() *// вывод содержимого буфера в файл (при буферизованном вводе-выводе)*

put() *// выводит символ в поток.*

write() *// выводит в поток указанное количество символов из буфера (неформатированный вывод)*

Пример использования потоков STL

#include <iostream>

#include <fstream>

using namespace std;

const char \*filename = "testfile2.txt";

int main() {

*// создание потока, открытие файла для записи в текстовом режиме,*

*// запись данных и закрытие файла.*

ofstream ostr;

ostr.open(filename);

if (ostr) {

for (int i = 0; i < 16; i++) {

ostr << i\*i << endl;

if (ostr.bad()) {

cerr << "Unrecoverable write error" << endl;

return 1;

}

}

ostr.close();

}

else {

cerr << "Output file open error \"" << filename << "\"" << endl;

return 1;

}

*// открытие файла (в конструкторе) для чтения в текстовом режиме,*

*// чтение данных, форматированный вывод на консоль, закрытие файла.*

int data;

int counter = 0;

ifstream istr(filename);

if (istr) {

while (!(istr >> data).eof()) {

if (istr.bad()) {

cerr << "Unrecoverable read error" << endl;

return 2;

}

cout.width(8);

cout << data;

if (++counter % 4 == 0) {

cout << endl;

}

}

istr.close();

}

else {

cerr << "Input file open error \"" << filename << "\"" << endl;

return 2;

}

return 0;

}

Взаимодействие потокового и традиционного ввода-вывода

Апологеты C++ рекомендуют использовать для ввода-вывода только потоки STL и отказаться от использования традиционного ввода-вывода в духе C. Однако, ничто не мешает, по крайней мере пока, использовать традиционную систему ввода-вывода. Более того, предусмотрена специальная функция для синхронизации ввода-вывода, выполненного посредством потоков и посредством старых функций.

#include <iostream>

bool sync\_with\_stdio(bool sync = true);

Заключение

Какой механизм использовать — вопрос предпочтений программиста, если работодателем явно не предписано использование конкретного механизма. В любом случае для физического ввода-вывода используются вызовы операционной системы. Всё остальное — обёртка, набор более или менее удобных функций или классов для взаимодействия с ОС.

Использование механизма потоков считается более безопасным. Но, как известно, плохую программу можно написать на любом языке программирования. Это также относится и к использованию библиотек.

**Составной оператор** — это группа операторов, отделённых друг от друга точкой с запятой, начинающихся с открывающей фигурной скобки { и заканчивающихся закрывающейся фигурной скобкой }:

{

оператор\_1;

...

оператор\_n;

}

Транслятор воспринимает составной оператор как одно целое.

Рассмотрим операторы языка С++, реализующие основные конструкции алгоритма.