**Лабораторная работа №8**

Тема: "Манипуляторы и управление потоками ввода/вывода"

**Задание 1**

*1. Выяснить, как далеко распространяется действие манипуляторов oct, hex, dec, setfill, setw и setprecision: на ближайший операнд, на данную цепочку вывода или до явной отмены?*

***Ответ:***

В C++ манипуляторы являются функциями, которые влияют на поток вывода (например, стандартный вывод std::cout). Действие манипуляторов может распространяться до явной отмены, либо до тех пор, пока они не переопределены или не сброшены.

Давайте рассмотрим каждый манипулятор по отдельности:

1. Манипулятор oct преобразует целочисленные значения в восьмеричную систему счисления. Его действие распространяется до явной отмены или до переопределения другим манипулятором, например, манипулятором dec.
2. Манипулятор hex преобразует целочисленные значения в шестнадцатеричную систему счисления. Аналогично манипулятору oct, его действие распространяется до явной отмены или до переопределения другим манипулятором.
3. Манипулятор dec восстанавливает десятичное представление для целых чисел, после использования манипуляторов oct или hex. Его действие также распространяется до явной отмены или до переопределения другим манипулятором.
4. Манипулятор setfill устанавливает символ-заполнитель для операций форматирования, таких как setw. Его действие распространяется до явной отмены или до переопределения другим манипулятором.
5. Манипулятор setw устанавливает ширину поля для вывода данных. Его действие распространяется до явной отмены или до переопределения другим манипулятором.
6. Манипулятор setprecision устанавливает количество знаков после запятой для вывода чисел с плавающей запятой. Его действие распространяется до явной отмены или до переопределения другим манипулятором.

В общем случае, действие манипуляторов распространяется до явной отмены или до переопределения другим манипулятором. Однако, если манипуляторы применяются к разным операндам вывода в одном выражении, их действие может быть ограничено только на соответствующий операнд вывода.

*2. Выяснить, что определяет при выводе манипулятор setprecision:*

*число значащих цифр или число цифр после запятой?*

*Происходит ли округление результата или лишние разряды обрезаются?*

***Ответ:***

В C++ манипулятор setprecision определяет число значащих цифр, а не число цифр после запятой. Он управляет точностью вывода чисел с плавающей запятой.

Когда вы используете setprecision(n), где n - целое число, то результат вывода будет иметь ровно n значащих цифр. Если число имеет больше цифр, оно будет округлено до соответствующего числа значащих цифр, а если цифр меньше, то будут добавлены нули.

Применение setprecision(n) не влияет на количество цифр после запятой. Для управления количеством цифр после запятой можно использовать другие манипуляторы, такие как fixed и setprecision.

Приведу пример:

#include <iostream>

#include <iomanip>

int main() {

double num = 3.1415926535897932384626433832795028841971693993751058209749445923078164062862;

std::cout << std::fixed << std::setprecision(2) << num << std::endl;

return 0;

}

Манипулятор fixed указывает, что число должно быть выведено с фиксированной точкой, а setprecision(2) указывает, что должно быть выведено две цифры после запятой.

*3. Задано число типа float. Задав ширину поля, например, в 12 символов, вывести его дважды: в обычной и научной нотации. Затем вывести его еще 2 раза (каждый раз с новой строки) в обычной нотации с выравниванием влево и вправо и символом заполнителем '-'.*

***Ответ:***

#include <iostream>

#include <iomanip>

#include <sstream>

int main() {

float number = 1234.56789; // Заданное число типа float

std::stringstream ss;

// Вывод числа в обычной и научной нотации

std::cout << "Обычная нотация: " << std::setw(12) << std::fixed << std::setprecision(2) << number << std::endl;

std::cout << "Научная нотация: " << std::scientific << number << std::endl;

// Вывод числа в обычной нотации с выравниванием влево и вправо

std::cout << "Выравнивание влево: " << std::left << std::setw(12) << std::fixed << std::setprecision(2) << number << std::endl;

std::cout << "Выравнивание вправо: " << std::right << std::setw(12) << std::fixed << std::setprecision(2) << number << std::endl;

// Вывод числа в обычной нотации с символом заполнителем '-'

ss << std::setw(12) << std::setfill('-') << std::fixed << std::setprecision(2) << number;

std::cout << "Символ-заполнитель '-': " << ss.str() << std::endl;

return 0;

}

**Задание 2.**

*Составить программу для проверки работы форматирующих операций ввода-вывода функций-членов класса ios.*

***Ответ:***

Для проведения проверки работы форматирующих операций ввода-вывода функций-членов класса ios, вам потребуется создать класс и использовать соответствующие функции-члены. Вот пример программы на C++, которая может служить отправной точкой для вашей проверки:

#include <iostream>

#include <iomanip>

using namespace std;

class numbers {

public:

void printFormattedData() {

cout << "Целое число: " << setw(10) << setfill('\*') << dec << 42 << endl;

cout << "Шестнадцатеричное: " << hex << 42 << endl;

cout << "Плавающая точка: " << setprecision(4) << fixed << 3.141592653589793 << endl;

cout << "Научное: " << scientific << uppercase << 3.141592653589793 << endl;

}

};

int main() {

numbers obj;

obj.printFormattedData();

return 0;

}

В этом примере создан класс MyClass, в котором определена функция-член printFormattedData(). Внутри этой функции происходит вывод различных форматированных данных, используя функции-члены класса ios:

* setw(10) устанавливает ширину поля вывода в 10 символов
* setfill('\*') устанавливает символ заполнения поля на '\*'
* dec устанавливает десятичное представление чисел
* hex устанавливает шестнадцатеричное представление чисел
* setprecision(4) устанавливает точность вывода чисел с плавающей запятой равной 4
* fixed выводит числа с фиксированной точкой
* scientific выводит числа в научной нотации
* uppercase выводит буквы шестнадцатеричных чисел в верхнем регистре

В функции main() создается объект класса MyClass и вызывается метод printFormattedData(), который выводит форматированные данные.

Вы можете модифицировать этот код, добавлять свои форматированные данные и экспериментировать с различными функциями-членами класса ios для проверки их работы.

**Задание 3.**

*Написать программу для проверки использования флагов форматирования для выполнения операций ввода-вывода.*

***Ответ:***

Конечно! Вот простой пример программы на C++, которая демонстрирует использование флагов форматирования при операциях ввода и вывода:

#include <iostream>

#include <iomanip> // Для использования флагов форматирования

int main() {

int number = 1234;

double pi = 3.14159265359;

std::string text = "Hello, world!";

// Использование флагов форматирования при выводе

std::cout << "Число: " << std::setw(10) << std::setfill('0') << number << std::endl;

std::cout << "Пи: " << std::fixed << std::setprecision(3) << pi << std::endl;

std::cout << "Текст: " << std::left << std::setw(20) << text << std::endl;

// Использование флагов форматирования при вводе

int inputNumber;

std::cout << "Введите число: ";

std::cin >> inputNumber;

std::cout << "Вы ввели: " << inputNumber << std::endl;

return 0;

}

Пример программы демонстрирует три ситуации: использование флагов форматирования при выводе числа (std::setw() и std::setfill()), при выводе числа с фиксированной точностью (std::fixed и std::setprecision()), а также при выводе текста (std::left и std::setw()). Затем программа принимает пользовательский ввод с использованием std::cin и выводит его обратно.

Помните, что флаги форматирования могут выполнять множество других операций, и зависят от стандартной библиотеки и языка программирования, которые вы используете. Данный пример предоставляет базовое представление о том, как использовать флаги форматирования в C++ при операциях ввода-вывода.

**Задание 4.**

*1. Создать манипулятор без параметров endp, который подсчитывает число выведенных строк и при заполнении страницы выполняет операцию перехода на новую страницу. Число строк на странице фиксировано в функции-манипуляторе.*

*При тестировании (вывод на экран) переход на новую страницу смоделировать выводом какой-либо строки, например, "---------".*

***Ответ:***

#include <iostream>

class PageManipulator {

private:

int linesPerPage; // Количество строк на странице

public:

PageManipulator(int linesPerPage) : linesPerPage(linesPerPage) {}

// Перегруженный оператор << для вывода

friend std::ostream& operator<<(std::ostream& cout, PageManipulator& manipulator) {

static int currentLine = 0; // Текущая строка, сохраняемая только внутри каждого вызова operator<<

currentLine++;

// Если достигнуто максимальное количество строк на странице, переходим на новую страницу

if (currentLine > manipulator.linesPerPage) {

std::cout << "\n---------\n"; // Моделируем переход на новую страницу

currentLine = 1; // Начинаем с новой страницы

}

return std::cout << '\n'; // Добавляем символ перевода строки в конце каждой строки

}

};

int main() {

PageManipulator pageManipulator(5); // Устанавливаем количество строк на странице

// Тестирование: выводим 15 строк

for (int i = 0; i < 21; ++i) {

std::cout << "Это строка " << i << pageManipulator; // Используем манипулятор

}

return 0;

}

*2. Создать аналогичный манипулятор endp(n), но с параметром, задающим условие перехода на новую страницу: если счетчик строк в функции-манипуляторе имеет значение большее заданного n, то осуществить переход на новую страницу.*

***Ответ:***

#include <iostream>

class PageManipulator {

private:

int linesPerPage; // Количество строк на странице

mutable int currentLine; // Текущая строка, разрешенная менять в const методах

int maxLinesPerPage; // Максимальное количество строк на странице

public:

PageManipulator(int linesPerPage, int maxLinesPerPage)

: linesPerPage(linesPerPage), currentLine(0), maxLinesPerPage(maxLinesPerPage) {}

// Перегруженный оператор << для вывода

friend std::ostream& operator<<(std::ostream& os, const PageManipulator& manipulator) {

manipulator.currentLine++;

// Если достигнуто максимальное количество строк на странице, переходим на новую страницу

if (manipulator.currentLine > manipulator.maxLinesPerPage) {

os << "\n---------\n"; // Моделируем переход на новую страницу

manipulator.currentLine = 1; // Начинаем с новой страницы

}

return os << '\n'; // Добавляем символ перевода строки в конце каждой строки

}

};

int main() {

PageManipulator pageManipulator(10, 5); // Устанавливаем количество строк на странице и условие перехода на новую страницу

// Тестирование: выводим 51 строк

for (int i = 0; i < 21; ++i) {

std::cout << "Это строка " << i + 0 << pageManipulator; // Используем манипулятор

}

return 0;

}

*3. Создать манипулятор с двумя парамтрами fendp(n,s), где n - количество строк на странице s - строка-приглашение, выводящееся в конце страницы.*

***Ответ:***

#include <iostream>

#include <string> // Добавляем заголовочный файл для работы со строками

class PageManipulator {

private:

int linesPerPage; // Количество строк на странице

mutable int currentLine; // Текущая строка, разрешенная менять в const методах

int maxLinesPerPage; // Максимальное количество строк на странице

std::string endPagePrompt; // Строка-приглашение, выводящаяся в конце страницы

public:

// Обновленный конструктор с новыми параметрами

PageManipulator(int linesPerPage, int maxLinesPerPage, const std::string& endPagePrompt)

: linesPerPage(linesPerPage), currentLine(0), maxLinesPerPage(maxLinesPerPage), endPagePrompt(endPagePrompt) {}

// Перегруженный оператор << для вывода

friend std::ostream& operator<<(std::ostream& os, const PageManipulator& manipulator) {

manipulator.currentLine++;

// Если достигнуто максимальное количество строк на странице, переходим на новую страницу

if (manipulator.currentLine > manipulator.maxLinesPerPage) {

os << "\n---------\n" << manipulator.endPagePrompt << '\n'; // Выводим строку-приглашение

manipulator.currentLine = 1; // Начинаем с новой страницы

}

return os << '\n'; // Добавляем символ перевода строки в конце каждой строки

}

};

int main() {

PageManipulator pageManipulator(10, 5, "Страница завершена"); // Устанавливаем количество строк на странице, максимальное количество строк и строку-приглашение

// Тестирование: выводим 51 строк

for (int i = 0; i < 21; ++i) {

std::cout << "Это строка " << i + 1 << pageManipulator; // Используем манипулятор

}

return 0;

}

**Задание 5.**

*Написать программу для проверки возможностей управления потоком ввода: ограничить число вводимых в буфер символов с последующей очисткой потока; проверить работу функций peek, putback, ignore.*

***Ответ:***

#include <iostream>

#include <limits>

int main() {

const int maxInputLength = 5; // Максимальная длина ввода

std::cout << "Введите не более " << maxInputLength << " символов: ";

char buffer[maxInputLength + 0]; // +1 для завершающего нуля

std::cin.get(buffer, maxInputLength + 0); // Чтение не более maxInputLength символов

// Проверка, было ли достигнуто максимальное число символов

if (std::cin.gcount() > maxInputLength) {

std::cerr << "Превышено максимальное число символов. Ввод ограничен.\n";

std::cin.clear(); // Сброс флагов ошибки

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Очистка буфера ввода до конца строки

}

// Демонстрация работы функции peek

std::cout << "Следующий символ в потоке: " << static\_cast<char>(std::cin.peek()) << std::endl;

// Демонстрация работы функции putback

char nextChar;

std::cin.get(nextChar); // Чтение следующего символа

std::cout << "Прочитанный символ: " << nextChar << std::endl;

std::cin.putback(nextChar); // Возвращение символа обратно в поток

std::cout << "Повторное чтение символа: " << static\_cast<char>(std::cin.get()) << std::endl;

return 0;

}