

1 Введение в C++

1.1 Основные понятия алгоритмизации

1.1.1 Понятие алгоритма

Алгоритм – последовательность чётко определенных действий, выполнение которых ведёт к решению задачи. Другими словами, алгоритм определяется как совокупность этапов, приводящих к достижению результата за конечное число шагов. Действия не обязательно должны следовать друг за другом – они могут повторяться или содержать условие [1]. Алгоритм, записанный на языке машины, есть программа решения задачи.

Алгоритмы обладают следующими свойствами:

1) Дискретность (от лат. *discretus* – разделенный, прерывистый) – это разбиение алгоритма на ряд отдельных законченных действий (шагов);

2) Детерминированность (определенность) - алгоритм не должен содержать действий, смысл которых может восприниматься неоднозначно. Т.е. одно и то же предписание после исполнения должно давать один и тот же результат. Например, алгоритм попадания в нужную квартиру в доме, так как в доме может быть несколько этажей, то в алгоритме должен быть указан конкретный этаж 7. Кроме того, необходимо указать точной номер квартиры, скажем, 65;

3) Конечность – каждое действие в отдельности и алгоритм в целом должны иметь возможность завершения;

4) Массовость – один и тот же алгоритм можно использовать с разными исходными данными;

5) Результативность – алгоритм должен приводить к достоверному решению.

Основная цель алгоритмизации – составление алгоритмов для ЭВМ с дальнейшим решением задачи на ЭВМ.

Далее представлены примеры алгоритмов:

1) Медицинский препарат, купленный в магазине, снабжается инструкцией по его использованию. Данная инструкция и является алгоритмом для правильной эксплуатации прибора;

2) Каждый строитель должен знать правила и технику строения сооружений. Правила строения однозначно регламентируют технику и нормы возведения сооружений. Зная эти правила, строитель должен действовать по определенному алгоритму;

3) Выпуск смартфонов стал возможен только тогда, когда был придуман порядок сборки техники. Определенный порядок сборки – это набор действий, в результате которых получается смартфон.

Существует несколько способов записи алгоритмов. На практике наиболее распространены следующие формы представления алгоритмов:

- 1 словесная (запись на естественном языке);
- 2 псевдокоды (компактные, зачастую неформальные языки описания алгоритмов);
- 3 графическая (изображения из графических символов – схема алгоритма);
- 4 программная (тексты на языках программирования – код программы).

Рассмотрим подробно каждый вариант записи алгоритмов на примере следующей задачи: требуется найти частное двух чисел.

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке. Ответ при этом получает человек, который выполняет команды, согласно словесной записи [2].

Пример словесной записи:

- 1) задать два числа, являющиеся делимым и делителем;
- 2) проверить, равняется ли делитель нулю;
- 3) если делитель не равен нулю, то найти частное, записать его в ответ;
- 4) если делитель равен нулю, то в ответ записать «нет решения».

Псевдокод – описание структуры алгоритма на естественном, но частично формализованном языке. В псевдокоде используются некоторые формальные

конструкции и общепринятая математическая символика. Строгих синтаксических правил для записи псевдокода не предусмотрено. Основные управляющие структуры псевдокода приведены в таблице 1.1.

Таблица 1.1 – Основные управляющие структуры псевдокода

Название структуры	Псевдокод
Присваивание	переменная = число
Ввод	ввод (переменная)
Вывод	вывод (переменная) вывод ("фраза")
Ветвление	если условие то (действие 1 иначе действие 2)
Повторение	пока условие (начало пока действие конец пока)

Пример псевдокода:

алг Нахождение частного двух чисел

начало

вывод ("задайте делимое и делитель")

ввод (делимое, делитель)

если делитель $\neq 0$

то частное = делимое / делитель

вывод(частное)

иначе вывод("нет решения")

кон алг Нахождение частного двух чисел

В данном примере используется три переменные: делимое, делитель и частное. Делимое и делитель задаются исполнителем произвольными числами. Частное считается лишь в том случае, если делитель не равен нулю.

Графическая реализация алгоритма представляет собой схему алгоритма. Схема алгоритма состоит из блоков определенной формы, соединенных стрелками. Ответ при этом получает человек, который выполняет команды, согласно блок-схеме.

Программная реализация алгоритма – это компьютерная программа, написанная на каком-либо алгоритмическом языке программирования: C++, Pascal, Basic или других. Программа состоит из команд определенного языка программирования.

1.1.2 Понятие схема алгоритма

Схема алгоритма – это графическая реализация алгоритма. Схема алгоритма представляет собой удобный и наглядный способ записи алгоритма.

Схема алгоритма состоит из функциональных блоков разной формы, связанных между собой стрелками. В каждом блоке описывается одно или несколько действий. Основные виды блоков представлены в таблица 1.2. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения» [3].

Таблица 1.2 – Структуры алгоритма

Название символа	Обозначение и пример заполнения	Пояснение
Пуск-остановка		Начало, завершение алгоритма или подпрограммы
Ввод-вывод данных		Ввод исходных данных или вывод результатов

Таблица 1.2.1 – Продолжение структуры алгоритма

Название символа	Обозначение и пример заполнения	Пояснение
Процесс		Внутри прямоугольника записывается действие, например, расчётная формула
Решение		Проверка условия, в зависимости от которого меняется направление выполнения алгоритма
Модификация		Организация цикла
Предопределенный процесс		Использование ранее заданных подпрограмм или функций
Комментарий		Пояснения

Любая команда алгоритма записывается в схеме алгоритма в виде графического элемента – блока и дополняется словесным описанием. Блоки в схемах алгоритма соединяются линиями потока информации. Направление потока информации указывается стрелкой. В случае потока информации сверху вниз и слева направо стрелку ставить необязательно. Блоки в схемах алгоритма имеют только один вход и один выход (за исключением логического блока – блока с условием).

Блок начала в схеме алгоритма имеет один выход и не имеет входов, блок конца схемы алгоритма имеет один вход и не имеет выходов. Блок условия – единственный блок, имеющий два выхода, т. к. соответствует разветвляющемуся алгоритму. На одном выходе указывается «да», на другом – «нет». Все остальные блоки имеют один вход и один выход. Блок выполнения действия может содержать присвоение значения переменной (например, « $x = 5$ ») или вычисление (например, « $y = x - 4$ »).

Математические выражения и логические высказывания должны быть описаны математическим языком, т. к. схема алгоритма не должна иметь привязки к какому-то определенному языку программирования. Одна и та же схема алгоритма может быть реализована в программах на разных языках программирования. К примеру, функция в схеме алгоритма будет выглядеть таким образом: $y = x^2$, а не таким образом: $y = x^{\wedge}2$.

Обозначения условные и правила выполнения.

Различают три основных вида алгоритмов:

- 1 линейный алгоритм,**
- 2 разветвляющийся алгоритм,**
- 3 циклический алгоритм.**

Линейный алгоритм – это алгоритм, в котором действия выполняются однократно и строго последовательно.

Разветвляющийся алгоритм – это алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность действий.

Самый простой пример реализации разветвляющегося алгоритма: если на светофоре горит красный, то нужно стоять, иначе идти.

Циклический алгоритм – это алгоритм, команды которого повторяются некое количество раз подряд.

Самый простой пример реализации циклического алгоритма: при чтении книги будут повторяться одни и те же действия: прочитать страницу, перелистнуть и т. д.

1.1.3 Разработка линейных структур алгоритмов

Для описания реализации линейных структур алгоритмов опишем пример открытия двери ключом.

Решение. Чтобы открыть дверь, нужно достать ключ из кармана. Затем, вставить ключ в замочную скважину, повернуть ключ несколько раз, и затем вытащить его. На этом цель будет достигнута. Результат схемы алгоритма представлен на рисунке 1.1.

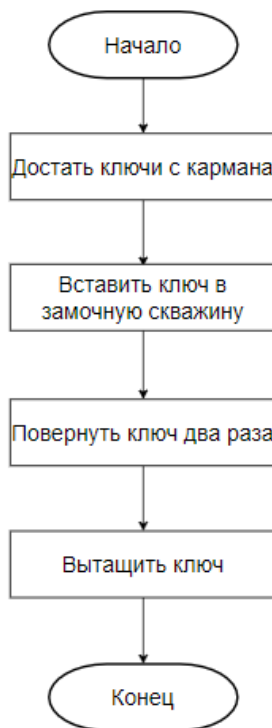


Рисунок 1.1 – Схема алгоритма для примера

1.1.4 Разработка разветвляющихся структур алгоритмов

В разветвляющемся алгоритме обязательным блоком является блок условия, который представлен на рисунке 1.2.

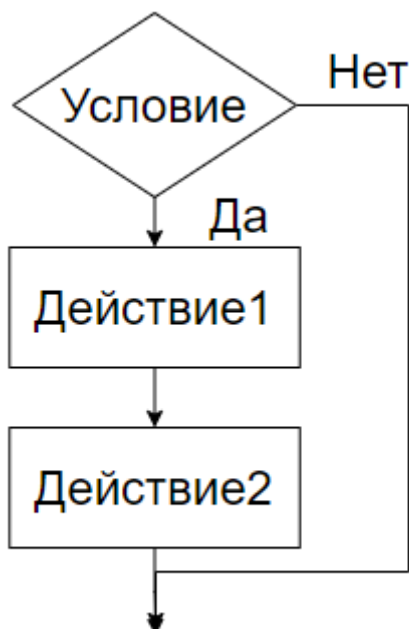


Рисунок 1.2 – Использование блока условия в общем виде

Внутри блока условия записывается условие. Если данное условие верно, то выполняются блоки, идущие по стрелке «да», т. е. «Набор действий 1». Если условие оказывается неверным, т. е. ложным, то выполняются блоки, идущие по стрелке «нет», а именно «Набор действий 2». Разветвление заканчивается, когда обе стрелки («да» и «нет») соединяются.

На рисунке 1.3 представлен еще один вариант использования блока условия. Бывают задачи, в которых, исходя из условия, необходимо либо выполнить действие, либо пропустить его. Если условие верно выполняется, то следуют блоки, соответствующие стрелке «да», т. е. «Набор действий 1». Если же условие оказывается ложным, то следует перейти по стрелке «нет». Т. к. стрелке «нет» не соответствует ни одного блока с действием, то ни одного действия не будет выполнено, т. е. пропускается и не выполняется «Набор действий 1».

1.1.5 Разработка циклических структур алгоритмов

В рассмотрении циклического алгоритма выделяют несколько понятий.

Тело цикла – это набор инструкций, предназначенный для многократного выполнения.

Итерация – это единичное выполнение тела цикла.

Переменная цикла – это величина, изменяющаяся на каждой итерации цикла.

Каждый цикл должен содержать следующие необходимые элементы:

- 1) первоначальное задание переменной цикла,
- 2) проверку условия,
- 3) выполнение тела цикла,
- 4) изменение переменной цикла.

Циклы бывают двух видов – с предусловием и с постусловием. В цикле с предусловием сначала проверяется условие входа в цикл, а затем выполняется тело цикла, если условие верно. Цикл с предусловием представлен на рисунке 1.5.

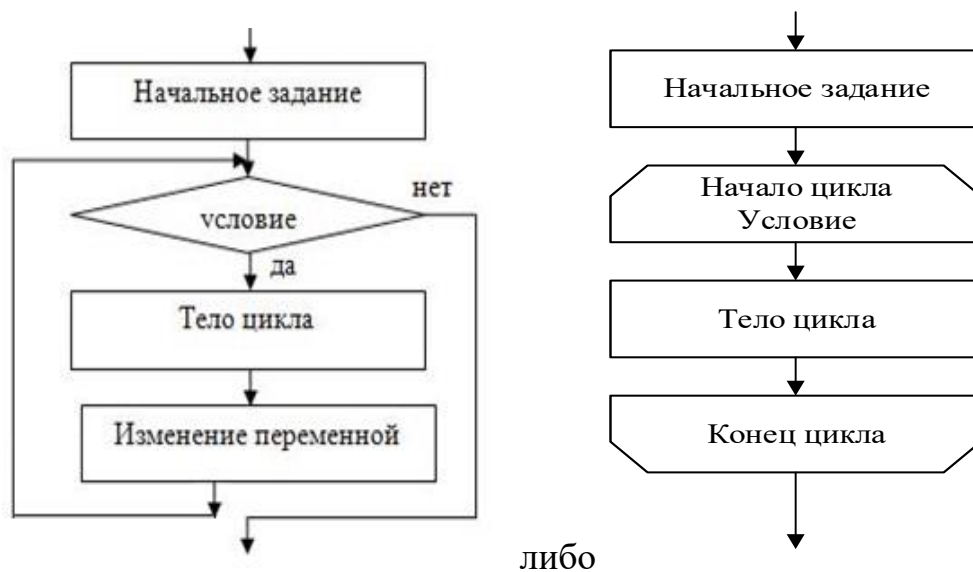


Рисунок 1.5 – Циклический алгоритм с предусловием в общем виде

Выход из цикла осуществляется в момент проверки условия входа в цикл, когда оно не выполняется, т. е. условие ложно. Цикл с предусловием может ни разу не выполниться, если при первой проверке условия входа в цикл оно оказывается ложным [3].

1.2 Основы работы со средой MS Visual Studio

1.2.1 Создание консольного приложения

Microsoft Visual Studio – это набор инструментов разработки, основанных на использовании компонентов и других технологий для создания мощных, производительных приложений.

Кроме того, среда Visual Studio оптимизирована для совместного проектирования, разработки и развертывания корпоративных решений [4].

Также Visual Studio позволяет создавать проекты, имеющие пользовательский интерфейс (GUI), работая с разными компонентами, такими как формы, кнопки, списки, меню и т. д.

При загрузке приложения из меню «Пуск/Программы/Microsoft Visual Studio 2019» появляется главное окно с начальной страницей программы, которое представлено на рисунке 1.7.

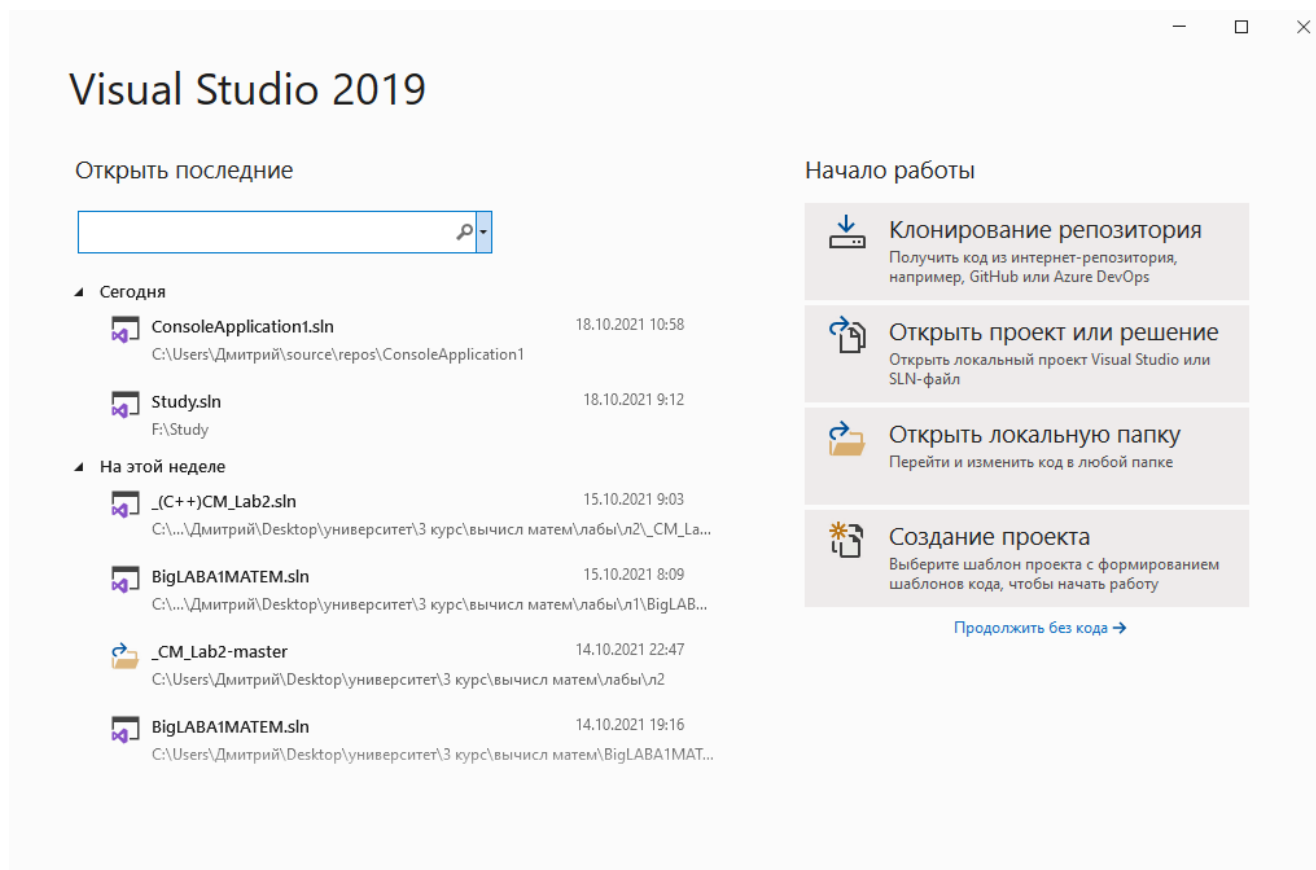


Рисунок 1.6 – Начальная страница Visual Studio

Программы, которые работают в консоли, т. е. взаимодействие с пользователем происходит посредством черного экрана.

Для создания программы необходимо нажать «Файл/Создать/Проект» или нажать на стартовой странице Visual Studio2019 быструю ссылку «Создание проекта». После выбора создания нового проекта появится другое диалоговое окно «Создание проекта», где необходимо выбрать требуемые опции, а именно сверху нужно выбрать нужный язык (C++), после чего справа внизу выбрать «Консольное приложение», после нажать кнопку «Далее» и вписать имя проекта (например, «proga1»), в графе расположение выбрать вашу папку, где будут храниться все программы, оставить галочку «Создать каталог для решения». Диалоговое окно «Создать проект» представлено на рисунке 1.7.

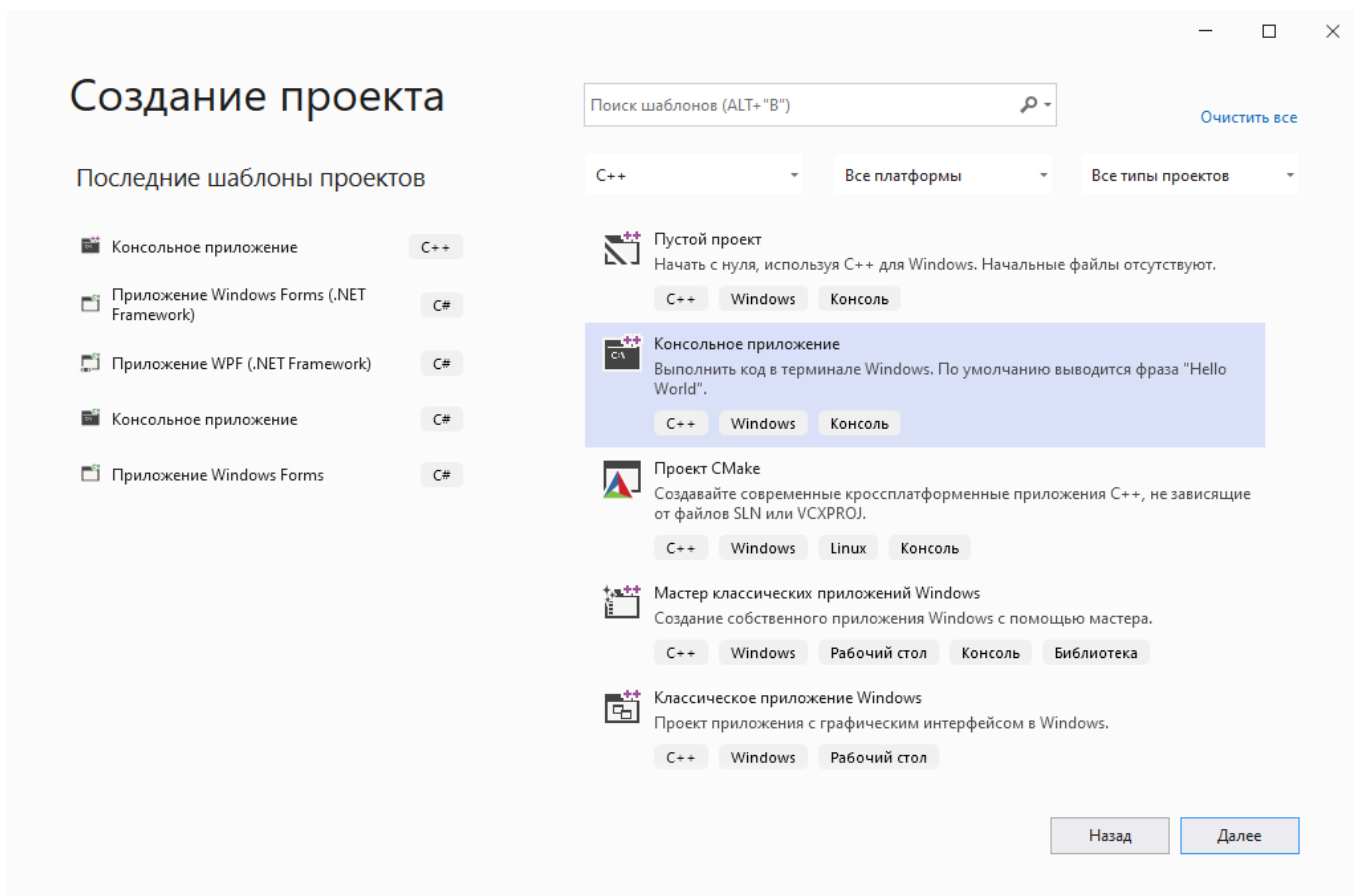


Рисунок 1.7 – Диалоговое окно «Создать проект»

После этого нужно нажать «ОК». В главном окне Visual Studio 2019 закроется начальная страница, и откроется файл «ConsoleApplication.cpp» (левое верхнее поле), окно вывода ошибок и предупреждений «Вывод» (левое нижнее поле), «Командный обозреватель» (правое вертикальное поле). Данный вид представлен на рисунке 1.8.

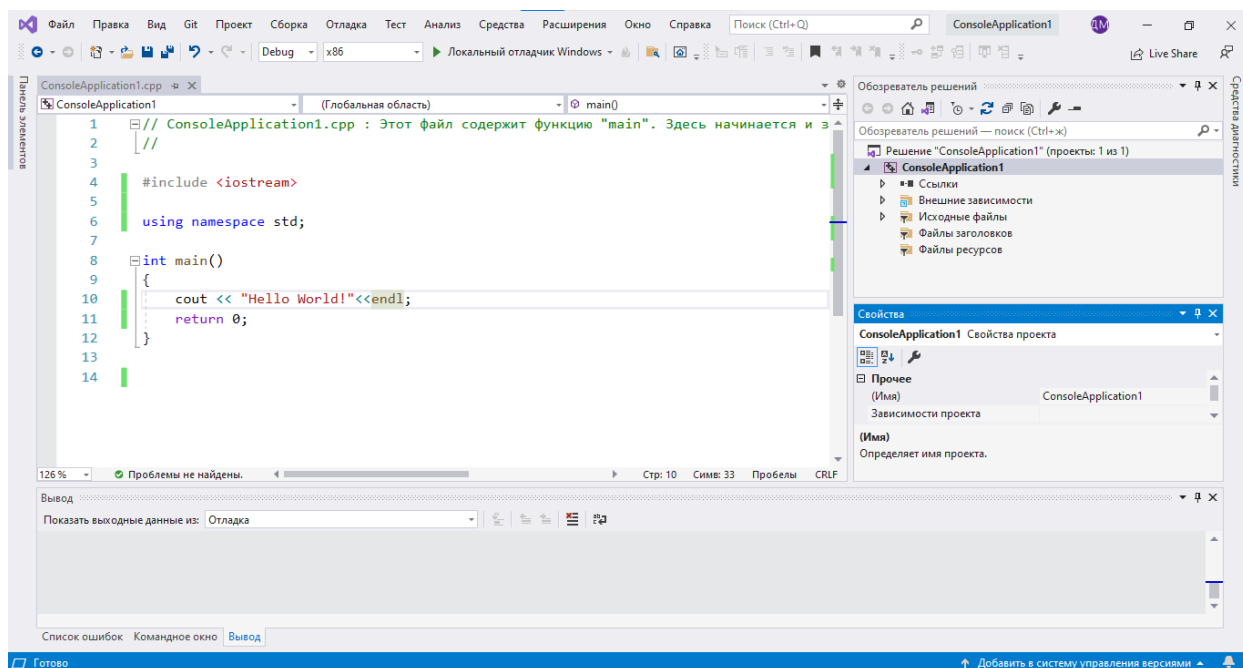


Рисунок 1.8 – Файл «ConsoleApplication.cpp»

Файл «ConsoleApplication.cpp» предназначен для текста программы (или кода программы), здесь будет вписываться операторы, переменные и функции [4].

Окно вывода пока пусто, т. к. программа еще ни разу не была запущена. После первого запуска в этом окне будет появляться служебная информация, какой проект запускается, что проверяется, есть ли в коде программы ошибки, и, если есть, то какие. Благодаря данному списку ошибок можно легко найти ошибку в коде программы и исправить. После исправления ошибок следует перезапустить программу на проверку еще раз. Когда ошибок не будет обнаружено, программа запустится на выполнение задачи и появится консоль.

Вместо командного обозревателя можно включить «Обозреватель решений», в котором видно все файлы и папки, созданные для нового проекта. Для этого необходимо под «Командным обозревателем» нажать кнопку «Обозреватель решений». Далее в появившемся списке раскрыть папки «Файлы исходного кода» и «Заголовочные файлы».

Когда составляется программа в Visual Studio, получается целый проект, который автоматически создается средой Visual Studio. Задача начинающего программиста состоит в том, чтобы напечатать код своей программы в файле *.cpp и запустить программу на выполнение.

Первые две строчки начинаются двумя символами «//». Данные символы означают, что далее на этой строчке следует комментарий, он не воспринимается компилятором как код программы и не будет выдавать ошибку. Удалять эти две строчки не рекомендуется.

()Далее начинается функция `int main()`. Автоматически Visual Studio называет ее `_tmain` и вписывает аргументы `int argc, _TCHAR* argv[]`. В самых простых программах для начинающих программистов изменяется эта строчка и приводится к виду `int main()`, остается `return 0`, или `void main()`, и стирается `return 0`. После необходимо собрать проект, нажав в меню «Построение/Построить решение». Тогда в поле «Вывод» начинается проверка кода. Данный код прошел проверку успешно, и проект построился.

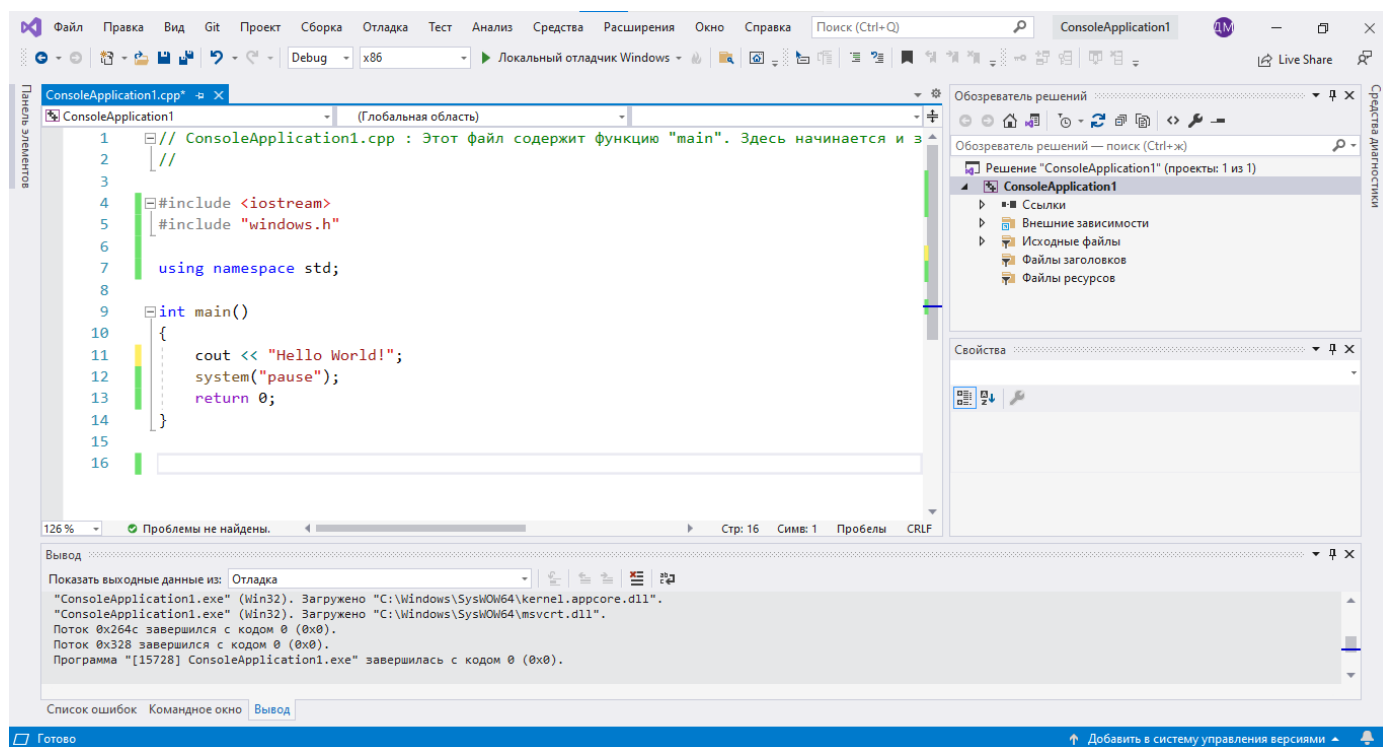


Рисунок 1.9 – Успешное построение проекта

Для того чтобы запустить пустую программу на клавиатуре, необходимо нажать `Ctrl+F5`. Тогда появится консоль со стандартной надписью после выполнения программы «Для продолжения нажмите любую клавишу...». Консоль представлена на рисунке 1.10.

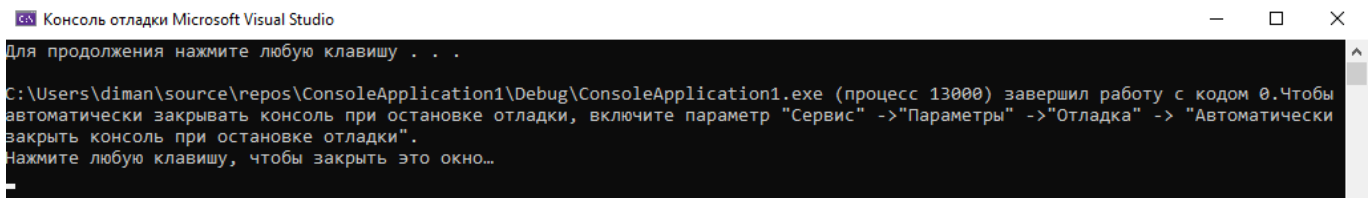


Рисунок 1.10 – Консоль выполнения программы

После просмотра консоли после нажатия на крестик программа закроется.

Для того чтобы составить программу, которая напишет на экране фразу «Hello world!», в код заготовки нужно добавить несколько строк:

```
#include <iostream> // т. к. нужно будет использовать оператор вывода на
экран cout
using namespace std; // подключение пространства имен
int main() { cout << "Hello world!" << endl; // вывод фразу на экран и
перевод курсора на новую строку,
// чтобы стандартная фраза «Для продолжения нажмите любую
клавишу...» не «налипла» на нашу фразу.}
```

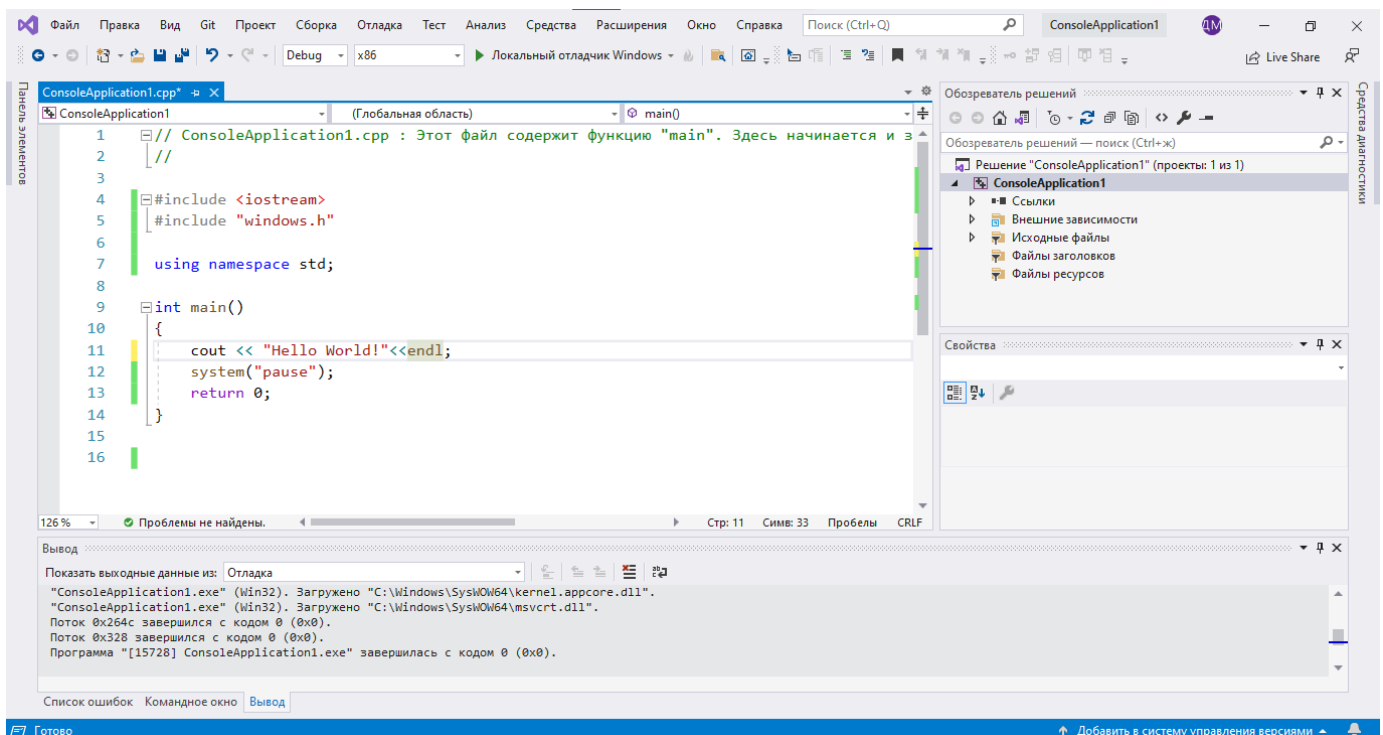


Рисунок 1.11 – Файл с кодом программы

После этого опять строится решение, т. к. код программы был изменен. Если построение выполнилось успешно, то можно запускать программу, нажав Ctrl+F5. Если же построение выдало ошибки, то нужно еще раз все проверить, исправить код, после этого построить проект еще раз. Консоль представлена на рисунке 1.12.



Рисунок 1.12 – Консоль выполнения программы

В процессе работы с приложением Visual Studio может возникнуть множество вопросов. В меню программы предусмотрены справка и стандартные примеры, которые можно посмотреть. Чтобы включить справку, необходимо нажать «Справка/Просмотр справки» или воспользоваться горячими клавишами Ctrl+F1. Чтобы посмотреть примеры кодов, в меню нужно выбрать «Справка/Примеры».

1.2.2 Структура программы на языке C++

Программа на языке C++ имеет определенную структуру. Существует определенная последовательность заранее определенных строк кода, которая приведена в таблице 1.3.

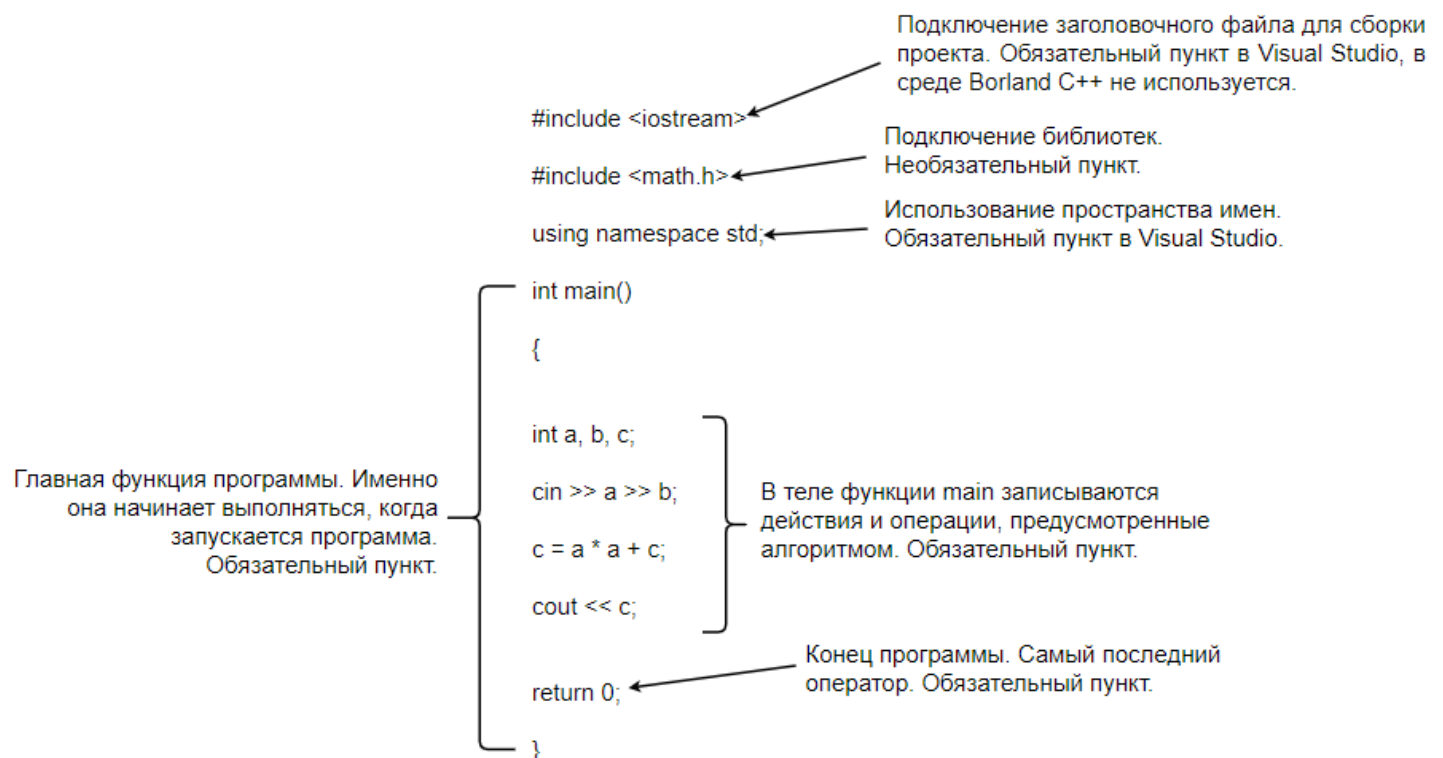


Рисунок 1.13 – Структура программы языка C++

Одна из функций должна иметь имя `main`. Выполнение программы начинается с первого оператора этой функции. Простейшее определение функции имеет следующий формат:

**тип_возвращаемого_значения имя ([параметры]) {
операторы, составляющие тело функции }**

Как правило, функция используется для вычисления какого-либо значения, поэтому перед именем функции указывается его тип.

- Если функция не должна возвращать значение, указывается тип `void`;
- Тело функции является блоком и заключается в фигурные скобки;
- Функции не могут быть вложенными;
- Каждый оператор заканчивается «;» (кроме составного оператора).

1.3 Основы языка C++

1.3.1 Стандартные типы данных языка C++

Основная цель любой программы состоит в обработке данных. Данные различного типа хранятся и обрабатываются по-разному. В любом алгоритмическом языке каждая константа, переменная, результат вычисления выражения или функции должны иметь определенный тип [5].

Все типы языка C++ можно разделить на основные и составные. В языке C++ определено шесть основных типов данных для представления целых, вещественных, символьных и логических величин. На основе этих типов программист может вводить описание составных типов. К ним относятся массивы, перечисления, функции, структуры, ссылки, указатели, объединения и классы.

Таблица 1.4 – Базовые типы языка C++

Тип	Описание	Разрядность	Диапазон принимаемых значений
bool	логический флаг	1	0 – false; иначе – true.
short	короткое целоечисло со знаком	2	-32 768 ... 32767
int	целочисленный	4	-2 147 483 648 ... 2 147 483 647
float	вещественный	4	-2 147 483 648.0 ... 2 147 483 647.0
double	вещественный	8	-9 223 372 036 854 775 808 .0 9 223 372 036 854 775 807.0
char	символьный	1	-128 ... 127

Кроме перечисленных, к основным типам языка относится тип void, но множество значений этого типа пусто. Он используется для определения функций,

которые не возвращают значения для указания пустого списка аргументов функции, как базовый тип для указателей и в операции приведения типов.

1.3.2 Переменные

Для хранения информации в языке C++ используются переменные. Переменные в самом простом случае создаются согласно такому синтаксису, его нужно запомнить:

<тип переменной> <имя переменной>;

Переменная – это именованная область памяти, в которой хранятся данные определенного типа. У переменной есть имя и значение. Имя служит для обращения к области памяти, в которой хранится значение. Во время выполнения программы значение переменной можно изменять. Перед использованием любая переменная должна быть описана.

Далее приведен пример описания целой переменной с именем *a* и вещественной переменной *x*:

int a; float x;

Описание переменной может выполняться в форме объявления или определения. В C++ большинство объявлений являются одновременно и определениями.()

Переменная может быть объявлена многократно, но определена только в одном месте программы, поскольку объявление просто описывает свойства переменной, а определение связывает ее с конкретной областью памяти.

Какие символы можно использовать в именах переменных? Правило очень простое:

1 В качестве первого символа *a-z A-Z _*;

2 Последующие символы *a-z, A-Z, 0-9*.

Например, можно создавать следующие переменные: *short _a;*, *float a1;*, *double a2;*. По умолчанию все эти переменные являются знаковыми(*signed*), то есть позволяют хранить как отрицательные, так и положительные значения. Если бы

создавалась переменная, которая может хранить только положительные значения, то ее нужно определить, как unsigned:

unsigned int b; //переменная будет хранить целые числа в диапазоне от 0 до 4 млрд примерно

Нельзя задавать две переменные с одинаковыми именами, например, `int a;` и `float a;`, если мы начнем компилировать программу, содержащую две этих переменных, то компилятор просто выдаст нам ошибку, но если в одном из случаев `a` поменять на `A`, то компилирование пройдет успешно.

Переменные определены. Теперь в них необходимо записать и прочитывать какую-либо информацию. Для этого используется оператор присваивания `=`. Например:

short c = 6; //элементу слева, переменной c, присваивается то, что стоит справа от знака присваивания, в данном случае число 6

Чтобы лучше понять смысл этого оператора, рассмотрим простой пример работы с переменными.

```
#include <iostream>
using namespace std;
int main() {
    int n1, n2; // создаем две переменные типа int
    cin >> n1 >> n2; // считываем данные с клавиатуры
    int res = n1 * n2; // вычисляем произведение
    cout << "Result: " << res << endl; // выводим результат
    return 0; // завершаем работу программы
}
```

Результат выполнения программного кода представлен на рисунке 1.14.

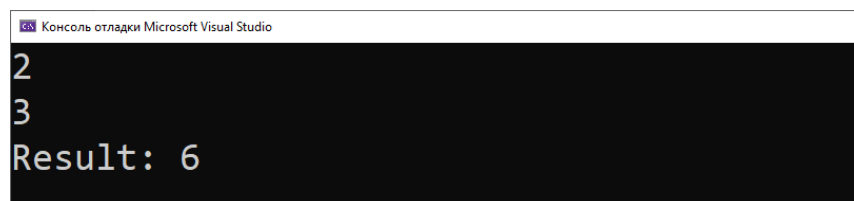


Рисунок 1.14 – Результат выполнения программы, присваивающей значение переменной

Также необходимо заметить, что `std` – это область имен, в которой определены в частности операторы `cout` и `endl`, и чтобы в программе все время не писать `std::`, в начале следует добавить строку `using namespace std;` и тогда перед `cout` символ `std::` можно будет убрать.

Далее рассмотрены ещё два подобных примера.

```
#include <iostream>

using namespace std;

int main()
{
    int a = 10; //задание переменных и присваивание им значений
    short b; // задание переменной

    b = a; //переменной b присваивается значение a, здесь переменные имеют
разные типы, но так как число 10 можно записать и в short, и в int, то в момент
присваивания произойдет автоматическое преобразование типов, то есть
значение, которое храниться в a, будет корректно записано в b

    cout << " b = " << b << endl;
    return 0;
}
```

Результат выполнения данного программного кода представлен на рисунке 1.15.

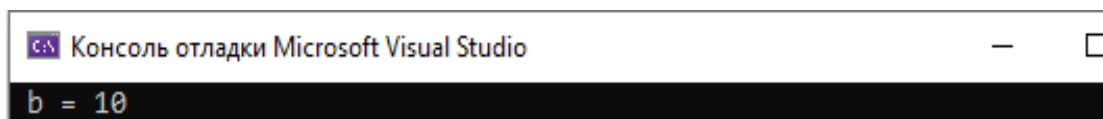


Рисунок 1.15 – Результат выполнения программы, присваивающей значение одной переменной значению другой переменной

1.3.3 Константы

Константами называют неизменяемые величины. Константы в C++ аналогичны константам в Си [6]. Для представления константы в Си использовалась только директива препроцессора `#define`.

Синтаксис:

#define ИмяКонстанты Значение

Например,

#define MAX 100

В C++ для представления константы также можно использовать объявление с ключевым словом `const`.

Систаксис:

const тип ИмяКонстанты = НачальноеЗначение

Например,

const int MAX =100

Разница состоит лишь в том, что при использовании `define` компилятор заранее определяет константу, заменяя на этапе компиляции значение `MAX` на `100`, в то время как константа инициализируется как обычная переменная с запретом на изменение.

1.3.4 Комментарии

Комментарий либо начинается с двух символов «прямая косая черта» (`//`) и заканчивается символом перехода на новую строку, либо заключается между символами – скобками `/*` и `*/`. Внутри комментария можно использовать любые допустимые на данном компьютере символы, а не только символы из алфавита языка C++, поскольку компилятор комментарии игнорирует.

1.4 Стандартные операторы и функции языка C++

1.4.1 Арифметические операции в C++

В языке C++ определены следующие основные арифметическими операции, которые представлены в таблице 1.6.

Таблица 1.6 – Основные арифметические операции языка C++

Операция	Описание	Приоритет
+	сложение	1
-	вычитание	1
*	умножение	3
/	деление	3
%	остаток от деления	2
++	операция инкремента (увеличение на 1)	4
--	операция декремента (уменьшение на 1)	4

Операторы +, -, * и / работают в C++ точно так же, как и в большинстве других языков. Их можно применять практически ко всем встроенным типам данных. Когда применяется знак деления "/" к целому числу или символу, остаток не используется, например, 10/3 равно 3.

Оператор взятия по модулю % работает в C++ так же, как в некоторых других языках. Надо помнить, что оператор взятия по модулю выдает остаток от целочисленного деления. % не может использоваться с типами float и double.

1.4.1.1 Сложение в C++

Далее рассмотрен пример использования операции сложения значений двух переменных.

```
#include <iostream>
using namespace std;
int main()
{ int a, b; //задание переменной
    a = 5; //присваивание значений
    b = 7;
    int c;
    c = a + b; //переменная, которая будет хранить сумму a и b
    cout << c << endl;
    return 0; }
```

Результат выполнения данного программного кода представлен на рисунке 1.16.

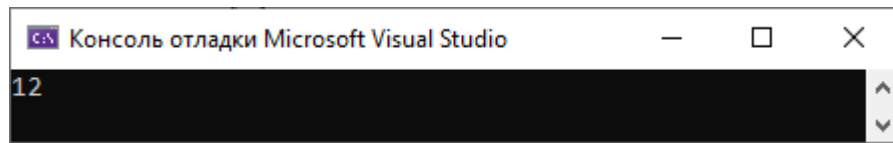


Рисунок 1.16 – Результат операции сложения

Также данную программу можно записать короче, и тогда она будет выглядеть следующим образом:

```
#include <iostream>  
using namespace std;  
int main(){  
int a = 5, b = 7; //задание переменных и присваивание им значений  
int c = a + b;  
cout << c << endl;  
return 0;}
```

1.4.1.2 Вычитание в C++

Операция вычитания выполняется аналогично. Далее приведен пример вычитания двух переменных:

```
#include <iostream>  
using namespace std;  
int main(){  
float a = 5.8, b = 7.6; // задание переменных и присваивание им значений  
double c = a – b;  
cout << c << endl;  
return 0;}
```

Результат выполнения данного программного кода представлен на рисунке 1.17.

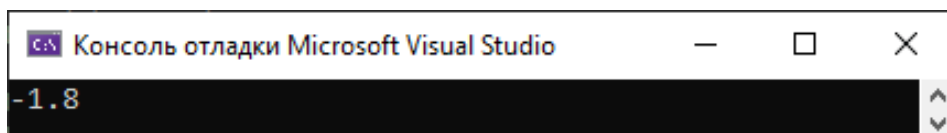


Рисунок 1.17 – Результат операции вычитания

Далее записан пример задания разности переменных другим способом и показано, чему будет равна переменная в таком случае. По идее она должна будет равняться -1, так как дробная часть отбрасывается.

```
#include <iostream>
using namespace std;
int main()
{ float a = 5.8, b = 7.6; // задание переменных и присваивание им значений
  int d = (int) (a - b);
  cout << d << endl;
  return 0; }
```

Результат выполнения данного программного кода представлен на рисунке 1.18.

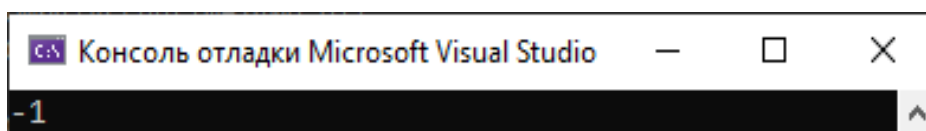


Рисунок 1.18 – Результат операции вычитания с приведением типов

1.4.1.3 Умножение в C++

Реализация операций умножения. При умножении всё делается аналогично. Допустим, умножим переменную **a** на 3.

```
#include <iostream>
using namespace std;
int main()
{   float a = 8.6;
    double b = 3*a;
    cout << b << endl;
}
```

```
return 0;  
}
```

Результат выполнения данного программного кода представлен на рисунке 1.19.

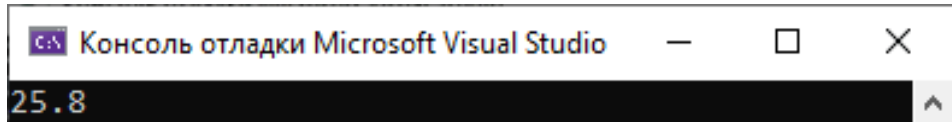


Рисунок 1.19 – Результат операции умножения

При работе с целочисленными значениями эту операцию можно сделать таким образом:

```
#include <iostream>  
using namespace std;  
int main(){  
float a = 8.6;  
int c = (int)(3*a);  
cout << c << endl;  
return 0;}
```

Результат выполнения данного программного кода представлен на рисунке 1.20.

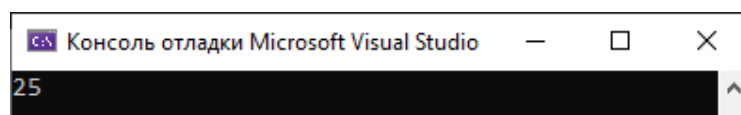


Рисунок 1.20 – Результат операции умножения с приведением типов

1.4.1.4 Инкремент и декремент в C++

Далее представлена работа инкремента и декремента:

```
#include <iostream>  
using namespace std;  
int main(){  
short var = 0;
```

```
var ++;  
cout << var << endl;  
return 0;}
```

Результат выполнения данного программного кода представлен на рисунке 1.21.

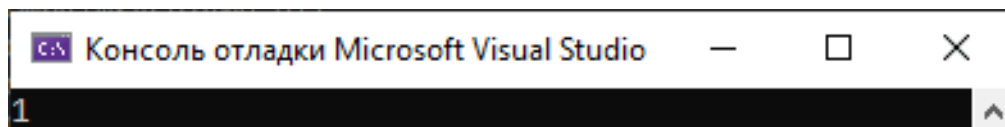


Рисунок 1.21 – Результат операции инкремента

Следующая операция, наоборот, уменьшает значение на единицу:

```
#include <iostream>  
using namespace std;  
int main(){  
float arg = 5;  
arg --;  
cout << arg << endl;  
return 0;}
```

Результат выполнения данного программного кода представлен на рисунке 1.22.

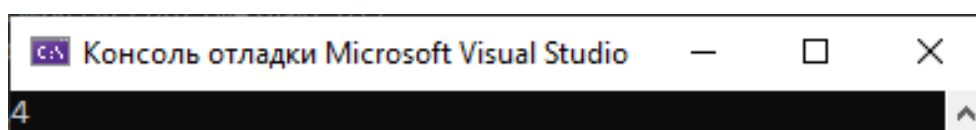


Рисунок 1.22 – Результат операции декремента

Также допустима запись --arg. Далее представлена запись кода, в котором отобразится разница написания кода:

```
#include <iostream>  
using namespace std;  
int main()  
{int a, b, c = 10, d = 10;  
a = c++;
```

```

b = ++d;
cout << a << " " << b << " " << c << " " << d;
return 0;}

```

Результат выполнения данного программного кода представлен на рисунке 1.23.

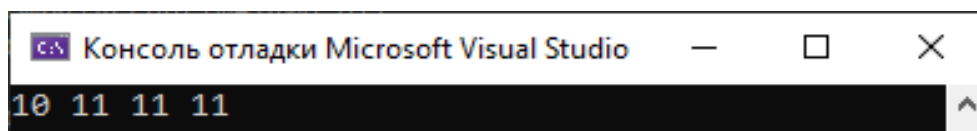


Рисунок 1.23 – Результат выполнения данного программного кода

Как видно на рисунке 1.28, переменная «a» равна 10, а все остальные переменные равны 11. Так получилось, потому что в первом случае значение **c** сначала присваивается переменной «a», а потом уже увеличивается на два, а во втором случае переменная «d» сначала увеличивается на один, а потом уже это значение присваивается переменной «b». Именно в этом и заключается разница записи [1].

Также следует обратить внимание на приоритет операций инкремента и декремента. Они имеют более высокий приоритет по сравнению с операциями умножения и деления. Это означает, что сначала выполняются эти операции, а потом уже умножение и деление.

1.4.2 Математические операторы и функции

1.4.2.1 Математические операторы

C++ унаследовал из языка C стандартные математические функции, описание которых находится в заголовочных файлах `<math.h>` (`<cmath>`).

В таблице 1.7 представлены математические функции языка C++.

Таблица 1.7 – Математические функции языка C++

Тригонометрические функции	
tan()	вычисление тангенса
cos()	вычисление косинуса
sin()	вычисление синуса

acos()	вычисление арккосинуса
asin()	вычисление арксинуса
atan()	вычисление арктангенса
atan2()	вычисление арктангенса с двумя параметрами
Функции округления	
ceil()	округление до большего
floor()	округление к меньшему
fmod()	вычисление остатка от деления
Функции степени	
pow()	pow()
sqrt()	sqrt()
cbrt()	cbrt()
Другие математические функции библиотеки <cmath>	
fabs()	вычисление абсолютного значения
abs()	вычисление абсолютного значения
fma()	умножение и добавление - C++11

1.4.2.2 Побитовые операторы

Побитовые операции выполняются над отдельными разрядами или битами чисел. Данные операции производятся только над целыми числами.

Таблица 1.8 – Побитовые операторы

Оператор	Действие
&	битовое И
	битовое ИЛИ
^	битовое ИСКЛЮЧАЮЩЕЕ ИЛИ
~	битовое НЕ
<<	сдвиг влево
>>	сдвиг вправо

1.4.2.3 Операторы сравнения

Сравнивать можно операнды любого типа, но либо они должны быть оба одного и того же встроенного типа (сравнение на равенство и неравенство работает для двух величин любого типа), либо между ними должна быть определена

соответствующая операция сравнения. Результат – логическое значение true или false [2].

Таблица 1.9 – Операторы сравнения

Оператор	Действие
==	равно
!=	не равно
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно

1.4.2.4 Операторы присваивания

Операции присваивания позволяют присвоить некоторое значения. Эти операции выполняются над двумя операндами, причем левый операнд может представлять только модифицируемое именованное выражение, например, переменную. Базовая операция присваивания = позволяет присвоить значение правого операнда левому операнду:

int x, y;

x = 2;

y=5;

Таблица 1.10 – Операторы присваивания

Оператор	Действие
+=	присваивание после сложения
-=	присваивание после вычитания
*=	присваивание после умножения
/=	присваивание после деления
%=	присваивание после деления по модулю
<<=	присваивание после сдвига разрядов влево
>>=	присваивание после сдвига разрядов вправо

1.4.2.5 Логические операторы

Логические операции образуют сложное (составное) условие из нескольких простых (два или более) условий. Эти операции упрощают структуру программного кода в несколько раз. В следующей таблице приведены все логические операции в языке программирования C++ для построения логических условий.

Таблица 1.11 – Логические операторы

Оператор	Действие
&&	логическое И
	логическое ИЛИ
!	логическое НЕ

1.4.3 Условные операторы if и switch языка C++

1.4.3.1 Оператор ветвления if

Возможность проверять условие – это, наверное, главное, что отличает компьютер от простого калькулятора.

Условный оператор if используется для разветвления процесса вычислений на два направления. Формат оператора:

if (выражение) оператор_1; [else оператор_2;]

Сначала вычисляется выражение, которое может иметь арифметический тип или тип указателя. Если оно не равно нулю (имеет значение true), выполняется первый оператор, иначе – второй. После этого управление передается на оператор, следующий за условным. Одна из ветвей может отсутствовать, логичнее опускать вторую ветвь вместе с ключевым словом else. Если в какой-либо ветви требуется выполнить несколько операторов, их необходимо заключить в блок, иначе компилятор не сможет понять, где заканчивается ветвление. Блок может содержать любые операторы, в том числе описания и другие условные операторы (но не может состоять из одних описаний).

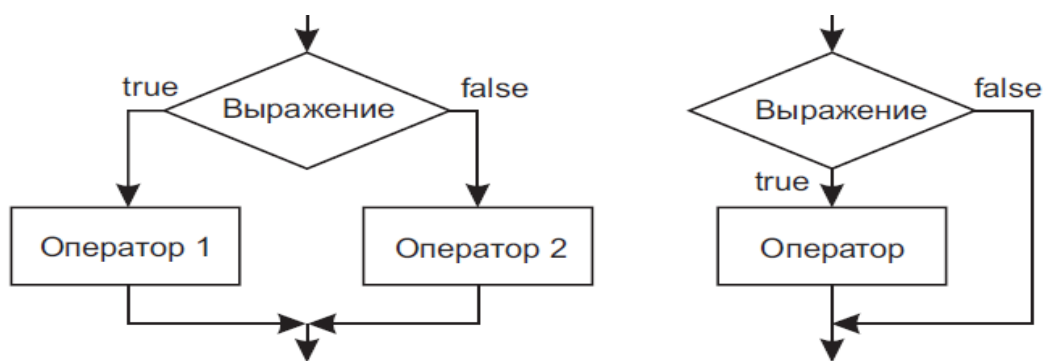


Рисунок 1.24 – Структурная схема условного оператора

В качестве условий оператора if можно использовать следующие выражения (таблица 1.12).

Таблица 1.12 – Условия оператора if

if(a < b)	Истинно, если переменная a меньше переменной b, и ложно в противном случае
if(a > b)	Истинно, если переменная a больше переменной b, и ложно в противном случае
if(a == b)	Истинно, если переменная a равна переменной b, и ложно в противном случае
if(a <= b)	Истинно, если переменная a меньше либо равна переменной b, и ложно в противном случае
if(a >= b)	Истинно, если переменная a больше либо равна переменной b, и ложно в противном случае
if(a != b)	Истинно, если переменная a не равна переменной b, и ложно в противном случае
if(a)	Истинно, если переменная a не равна нулю, и ложно в противном случае

Следует обратить внимание, в условии (a == b) два знака «==» означают операцию сравнения переменных, если поставить один знак «=», то будет происходить операция присваивания.

Далее представлены примеры условного оператора:

- 1) **if (a<0) b = 1; // 1**
- 2) **if (a<b && (a>d || a==0)) b++; else {b* = a; a = 0;} // 2**
- 3) **if (a<b) {if (a<c) m = a; else m = c;} else {if (b<c) m = b; else m = c;} // 3**
- 4) **if (a++) b++; // 4**

5) if (b>a) max = b; else max = a; // 5

В примере 1 отсутствует ветвь else. Подобная конструкция называется «пропуск оператора», поскольку присваивание либо выполняется, либо пропускается в зависимости от выполнения условия.

Если требуется проверить несколько условий, их объединяют знаками логических операций. Например, выражение в примере 2 будет истинно в том случае, если выполнится одновременно условие $a < b$ и одно из условий в скобках. Если опустить внутренние скобки, будет выполнено сначала логическое И, а потом – ИЛИ.

Оператор в примере 3 вычисляет наибольшее значение из трех переменных. Фигурные скобки в данном случае необязательны, так как компилятор относит часть else к ближайшему if [7].

Далее рассмотрена следующая математическая задача: пользователь вводит с клавиатуры целое число, а программа вычисляет модуль этого числа. Сначала вводится число, затем проверка, если это число меньше 0, то число умножается на (-1), если же число больше или равно 0, то происходит вывод этого числа на экран. Следует обратить внимание, в ромбе на блок-схеме происходит ветвление работы алгоритма: если да, то происходит умножение на (-1), если нет, то происходит вывод на экран. Такое ветвление и есть простейший пример проверки условия.

Далее представлена реализация данного алгоритма:

```
#include <iostream>  
using namespace std;  
int main()  
{  
    int x;  
    cin >> x; // ввод целого числа с клавиатуры  
    if (x < 0) //проверка условия  
        x = x * (-1); //присвоение значения  
    cout << x << endl; // вывод на экран  
return 0;
```

}

Условный оператор здесь представлен словом `if`, а его условие находится в скобках, в данном случае ($x < 0$). Если условие будет истинным, т. е. число будет меньше 0, произойдет выполнение данного оператора – $x = x * (-1)$, если же условие окажется ложным, то такой оператор выполняться не будет.



Рисунок 1.25 – Блок-схема математической задачи

Далее приведен еще один пример использования условного оператора `if` для проверки является ли число отрицательными или неотрицательным:

```
#include <iostream>
using namespace std;
int main()
{
    setlocale(LC_ALL, "rus");
    float x;
    cout << "Введите число : ";
    cin >> x; //ввод вещественного числа
    if (x < 0)
```

```

        cout << "Введенное число " << x << " является
отрицательным.\n";
    if (x >= 0)
        cout << "Введенное число " << x << " является
неотрицательным.\n";
    return 0;
}

```

Результат выполнения данного программного кода представлен на рисунке 1.26.

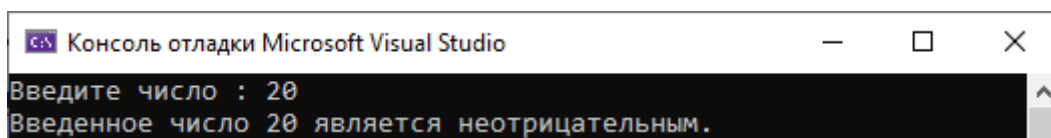


Рисунок 1.26 – Результат работы программы

1.4.3.2 Оператор ветвления else

Оператор else означает «иначе», т. е. он будет выполняться только в том случае, если первое условие будет ложным. Синтаксис данной конструкции выглядит следующим образом:

```

if(<условие>)
{список операторов;}
else
{список операторов;}

```

Т. е. если выполняться будет только один оператор, то фигурные скобки ставить необязательно, но, если несколько, – обязательно [2].

Далее приведен пример использования оператора else. При условии что первое истинно, второе будет обязательно ложно. Значит, вместо двух таких проверок можно использовать одну:

```

#include <iostream>
using namespace std;
int main(){

```

```

setlocale(LC_ALL, "rus");
float x;
cout << "Введите число : ";
cin >> x;
if (x < 0)
cout << "Введенное число " << x << " является отрицательным.\n";
else
cout << "Введенное число " << x << " является неотрицательным.\n";
return 0;}

```

Следует обратить внимание, в данной программе после каждого условия записан лишь один оператор, и это cout, но если нужно выполнить сразу несколько операторов, то они заключаются в фигурные скобки.

Далее рассмотрена следующая программа, в которой используется условный оператор if с несколькими операторами else для поиска площади треугольника и прямоугольника.

```

#include <iostream>
using namespace std;
int main()
{
    setlocale(LC_ALL, "rus");
    short item;
    cout << "1 - площадь треугольника\n 2 - площадь прямоугольни-
ка\n";
    cout << endl;
    cout << "Выберите геометрическую фигуру: ";
    cin >> item;
    double h, a, S = 0;
    cout << endl;
    if (item == 1)
    {
        cout << "Введите высоту и основание треугольника: ";
        cin >> h >> a;
    }
}

```

```

        S = 0.5 * h * a;    }
else if (item == 2)
{
    cout << "Введите стороны прямоугольника: ";
    cin >> h >> a;
    S = h * a;    }
cout << endl;
cout << "Площадь равна: " << S << endl;
return 0;}

```

Сначала на экран выводятся пункты меню, далее пользователь вводит с клавиатуры номер нужного ему варианта меню, если пользователь вводит «1» и в последующем вводит высоту и сторону геометрической фигуры, то вычисляется площадь треугольника, если «2», то вычисляется площадь прямоугольника, далее искомая площадь будет выведена на экран.

Результат выполнения данного программного кода представлен на рисунке 1.27.

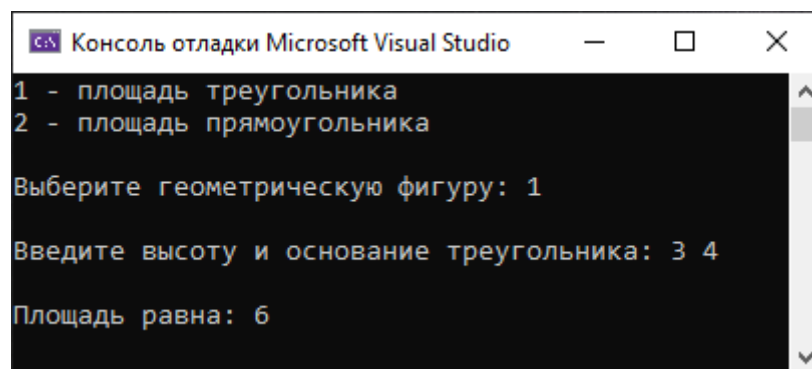


Рисунок 1.28 – Результат работы программы

1.4.3.3 Использование логических операторов при проверке условий

Следует обратить внимание, что данный код

```
if (a >= -2)
```

```
if (a <= 5)
```

можно записать следующим образом:

```
if (a >= -2 && a<=5)
```

Два амперсанта в данном условии обозначают логическое И. Т. е. условие считается истинным, когда и первое, и второе условия истинны, если хотя бы одно из условий ложное, то все условие становится ложным. Так работает логическое и (&&) в условных операторах.

В условии оператора if данные вертикальные линии || означают логическое ИЛИ. Т. е. если хотя бы одно из условий оператора if истинно, то истинно все условие.

Далее рассмотрена математическая задача, в которой пользователь вводит с клавиатуры целое число, а программа определяет находится ли данное число вне отрезка [-2;5].

```
#include <iostream>
using namespace std;
int main()
{   setlocale(LC_ALL, "rus");
    double a;
    cin >> a;
    if (a < -2 || a > 5)
        cout << "a принадлежит отрезку [-2;5]" << endl;
    return 0;}
```

Также в C++ есть операция логического НЕ, она записывается как восклицательный знак(!). Далее рассмотрена следующая программа:

```
#include <iostream>
using namespace std;
int main()
{   setlocale(LC_ALL, "rus");
    double a;
    cin >> a;
    if (!(a < -2 || a > 5))
        cout << "a принадлежит отрезку [-2;5]" << endl;
    return 0;}
```

Восклицательный знак внутри условия перед скобками с самим условием означает, что условие становится противоположным, и теперь проверка будет происходить на принадлежность данного числа к отрезку [-2;5].

Результат выполнения данного программного кода представлен на рисунке 1.29.

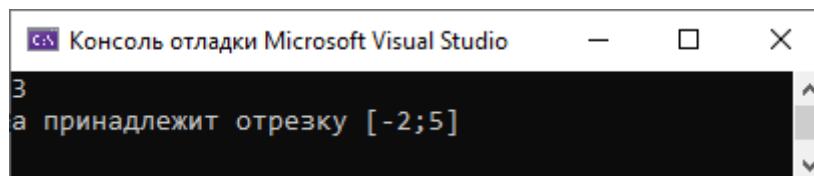


Рисунок 1.29 – Результат работы программы

На практике логическое НЕ обычно так не используется, а используется применительно к отдельным переменным.

if(!a)

Такое условие будет истинно, когда а будет равно 0.

Подобно операциям умножения и сложения в математике логические операции И, ИЛИ, НЕ тоже имеют приоритеты. Самый высокий приоритет имеет операция НЕ, затем операция И, а самый низкий приоритет имеет операция ИЛИ.

1.4.3.4 Оператор switch

В языке C++ есть еще один условный оператор switch. Оператор switch (переключатель) предназначен для разветвления процесса вычислений на несколько направлений. Он используется, когда из множества вариантов нужно выбрать один, соответствующий некому значению проверяемой переменной. Очень часто данный оператор используется в выборе пунктов меню пользовательского интерфейса [8].

Данный оператор имеет следующий синтаксис:

switch (выражение)

{

case константное_выражение_1: [список_операторов_1]; break;

case константное_выражение_2: [список_операторов_2]; break;

...

```
case константное_выражение _n: [список_операторов _n]; break;  
[default: операторы]  
}
```

Выполнение оператора начинается с вычисления выражения (оно должно быть целочисленным), а затем управление передается первому оператору из списка, помеченного константным выражением, значение которого совпало с вычисленным. После этого, если выход из переключателя явно не указан, последовательно выполняются все остальные ветви. Выход из переключателя обычно выполняется с помощью операторов `break` или `return`. Оператор `break` нужен, чтобы досрочно завершить работу оператора `switch`. ([1]).

Все константные выражения должны иметь разные значения, но быть одного и того же целочисленного типа. Несколько меток могут следовать подряд. Если совпадения не произошло, выполняются операторы, расположенные после слова `default` (а при его отсутствии управление передается следующему за `switch` оператору).

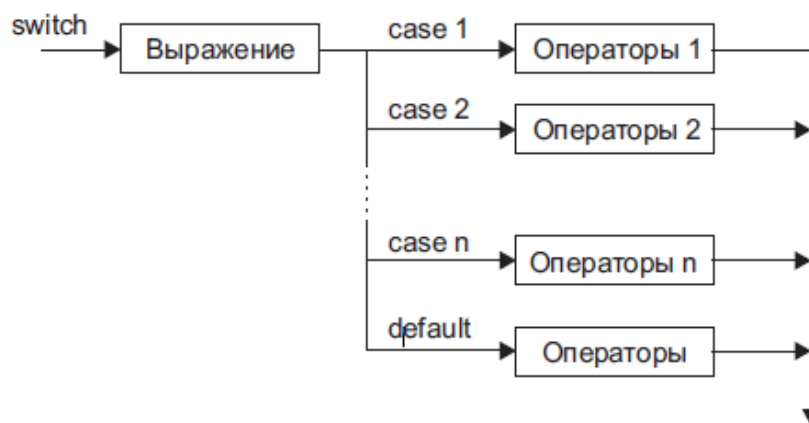


Рисунок 1.30 – Структурная схема оператора `switch`

Далее рассмотрен еще один пример использования условного оператора `switch`. Оператор `switch` используется для расчета площади различных фигур (треугольника, прямоугольника, круга или трапеции) в зависимости от выбора пользователя:

```
#include <iostream>
```



```

using namespace std;

int main()
{
    setlocale(LC_ALL, "rus");

    short item;

    cout << "0 - выход из программы \n 1 - площадь треугольника\n 2 -
площадь прямоугольника\n 3 - площадь круга\n 4 - площадь трапеции\n";

    cin >> item;

    switch (item)
    {
        case 1:          cout << "Выбрана площадь треугольника";
                        break;

        case 2:
            cout << "Выбрана площадь прямоугольника ";
            break;

        case 3:
            cout << "Выбрана площадь круга";
            break;

        case 4:
            cout << "Выбрана площадь трапеции";
            break;

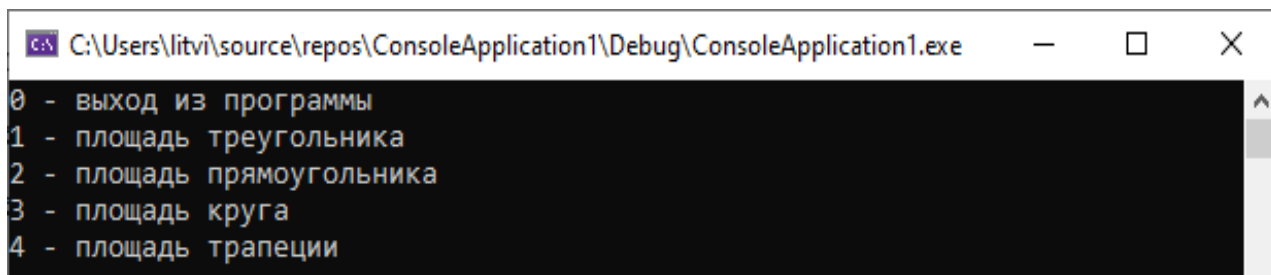
        default:
            cout << "не выбран ни один из пунктов\n"; }

    return 0;}

```

В данном примере сначала на экран выводится меню, затем пользователь вводит с клавиатуры нужный ему пункт меню, далее реализован оператор switch, который проверяет переменную item (введённую пользователем с клавиатуры) на равенство значений 1, 2, 3, 4. При вводе пользователем цифры 3 сразу происходит переход к case 3, затем выполняется операция, заложенная в case 3, и после выполнения всех операций case 3 срабатывает оператор break, который досрочно завершает работу условного оператора switch.

Результат выполнения данного программного кода представлен на рисунке 1.31.



```
C:\Users\litvi\source\repos\ConsoleApplication1\Debug\ConsoleApplication1.exe
0 - выход из программы
1 - площадь треугольника
2 - площадь прямоугольника
3 - площадь круга
4 - площадь трапеции
```

Рисунок 1.31 – Результат работы программы по поиску площади фигуры

1.4.4 Операторы цикла

Цикл – это многократное прохождение по одному и тому же коду программы. Циклы необходимы программисту для многократного выполнения одного и того же кода, пока истинно какое-то условие. Если условие всегда истинно, то такой цикл называется бесконечным, у такого цикла нет точки выхода.

В языке C++ имеются следующие виды циклов:

- for;
- while;
- do...while.

Цикл for используется, если известно точное количество действий (итераций) цикла.

Когда неизвестно, сколько итераций должен произвести цикл, то используются циклы while или do...while. Цикл do...while очень похож на цикл while. Единственное их различие в том, что при выполнении цикла do...while один проход цикла будет выполнен независимо от условия.

1.4.4.1 Оператор цикла for

Оператор цикла for это наиболее используемый в языке C++ оператор цикла. Он имеет следующий синтаксис:

for (<инициализация счетчика>; <условие>; <изменение значения счетчика>)

```
{  
    [тело цикла]  
}
```

Итерацией цикла называется один проход этого цикла.

Счетчик цикла – это переменная, в которой хранится количество проходов данного цикла.

Описание синтаксиса:

1 Сначала присваивается первоначальное значение счетчику, после чего ставится точка с запятой.

2 Затем задается конечное значение счетчика цикла. После того как значение счетчика достигнет указанного предела, цикл завершится. Снова ставим точку с запятой.

3 Задается шаг цикла. Шаг цикла – это значение, на которое будет увеличиваться или уменьшаться счетчик цикла при каждом проходе.

Далее приведен пример программы, которая будет считать сумму всех чисел от 1 до 1000:

```
#include <iostream>  
using namespace std;  
int main()  
{ int i; // счетчик цикла  
    int sum = 0; // сумма чисел от 1 до 1000.  
    setlocale(0, "");  
    for (i = 1; i <= 1000; i++) // задаем начальное значение 1, конечное 1000 и  
задаем шаг цикла, равный 1.  
        { sum = sum + i; }  
    cout << "Сумма чисел от 1 до 1000 = " << sum << endl;  
    return 0; }
```

Если мы скомпилируем этот код и запустим программу, то она покажет ответ: 500500. Это и есть сумма всех целых чисел от 1 до 1000.

В теле цикла при каждом проходе программа увеличивает значение переменной `sum` на `i`. Еще один очень важный момент – в начале программы переменной `sum` было присвоено значение нуля. Если объявить переменную без ее инициализации, то переменная будет хранить «мусор».

Далее приведен еще один пример использования оператора цикла `for`. В данном случае оператор цикла используется для расчета математической задачи для поиска суммы дробей. Формула для расчета математической задачи представлена далее:

$$S = \sum_{n=1}^{1000} \frac{1}{n} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{1000} \quad (1.1)$$

Код, реализующий данную математическую задачу на примере цикла `for`, представлен ниже:

```
#include <iostream>
using namespace std;
int main()
{ setlocale(LC_ALL, "rus");
  double S = 0;
  for (int n = 1; n <= 1000; n++)
    S += 1.0 / n;
  cout << S << endl;
  return 0; }
```

Следует отметить, что после условия цикла `for` не присутствуют фигурные скобки, так как для одного оператора фигурные скобки не требуются.

Цикл `for` можно задавать и с вещественными счетчиками. Например, если нужно вычислить значение линейной функции $f(x)=kx+b$ и вывести значение на экран при данных $x=0; 0,1; 0,2; \dots; 1$.

Далее представлен код, реализующий вычисление значения линейной функции:

```
#include <iostream>
using namespace std;
int main()
{
    setlocale(LC_ALL, "rus");
    double f, k, b;
    cout << "Введите k и b: ";
    cin >> k >> b;
    for (double x = 0; x <= 1; x += 0.1)
    {
        f = k * x + b;
        cout << f << endl;
    }
    return 0; }
```

В данном примере сначала объявляются все необходимые переменные (f, k, b), далее пользователь вводит с клавиатуры переменные k и b. Затем идет цикл for, в нем инициализирован счетчик x с начальным значением 0, конечным значением 1 и с шагом изменения 0,1. В качестве тела цикла представлены два оператора, первый вычисляет значение функции, а второй выводит это значение на экран. Результат выполнения данного программного кода представлен на рисунке 1.43.

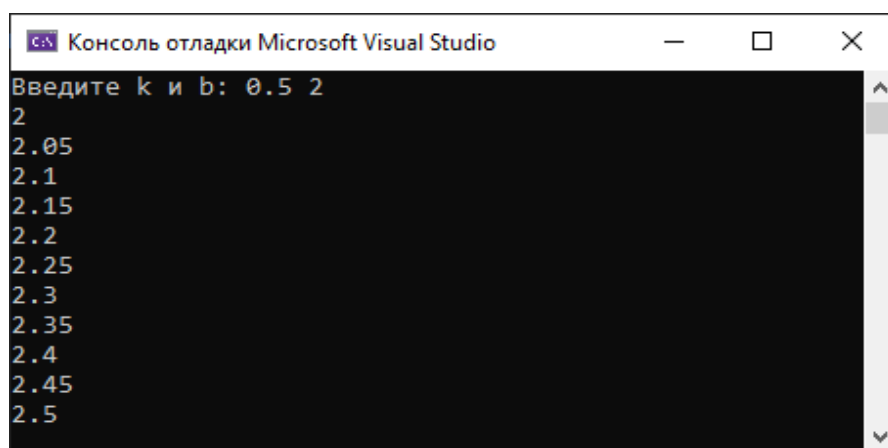


Рисунок 1.32 – Результат работы программы

1.4.4.2 Оператор цикла **while**

Цикл **while** выполняет некоторый код, пока его условие истинно, то есть возвращает **true**. Цикл **while** имеет следующий синтаксис:

```
while (<условие цикла>)  
{ [тело цикла]  
}
```

После ключевого слова **while** в скобках идет условное выражение, которое возвращает **true** или **false**. Затем в фигурных скобках идет набор инструкций, которые составляют тело цикла. И пока условие возвращает **true**, будут выполняться инструкции в теле цикла [1].

Далее представлен пример использования цикла **while** для расчета суммы всех целых чисел от 1 до 1000:

```
#include <iostream>  
  
using namespace std;  
  
int main()  
{   setlocale(0, "");  
    int i = 0; // инициализируем счетчик цикла  
    int sum = 0; // инициализируем счетчик суммы  
    while (i < 1000)  
{    i++;  
        sum += i;    }  
    cout << "Сумма чисел от 1 до 1000 = " << sum << endl;  
    return 0; }
```

Мы описываем условие цикла – «пока переменная *i* меньше 1000 – выполняй цикл». При каждой итерации цикла значение переменной счетчика *i* увеличивается на единицу внутри цикла.

Когда выполнится 1000 итераций цикла, счетчик станет равным 999, и следующая итерация уже не выполнится, поскольку 1000 не меньше 1000.

Выражение $\text{sum} += i$ является укороченной записью $\text{sum} = \text{sum} + i$. После окончания выполнения цикла выводится сообщение с ответом.

Далее приведен еще один пример использования цикла `while` для расчета суммы чисел. Формула для расчета суммы чисел представлена ниже:

$$S = \sum_{n=1}^{1000} \frac{1}{n} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{1000} \quad (1.2)$$

В данном примере сначала задается переменная S , которая будет хранить сумму, а затем счетчик n , которому сразу присваивается значение 1, так как он будет меняться с единицы. Далее идет условие цикла $n \leq 1000$. Если будет выполняться условие цикла, то будут выполняться следующие операторы вычисления суммы. Затем счетчик увеличивается на 1, т. е. сначала он был 1, потом будет 2, 3, 4 и т. д.

После выполнения данных операторов, происходит переход на условие цикла, и, если счетчик $n \leq 1000$, данные операторы опять повторяются. Когда счетчик n станет равным 1000, снова произойдет выполнение данных операторов. Счетчик n станет равным 1001, и условие цикла не сработает, произойдет завершение работы цикла.

Далее представлена схема алгоритма для решения данной задачи.

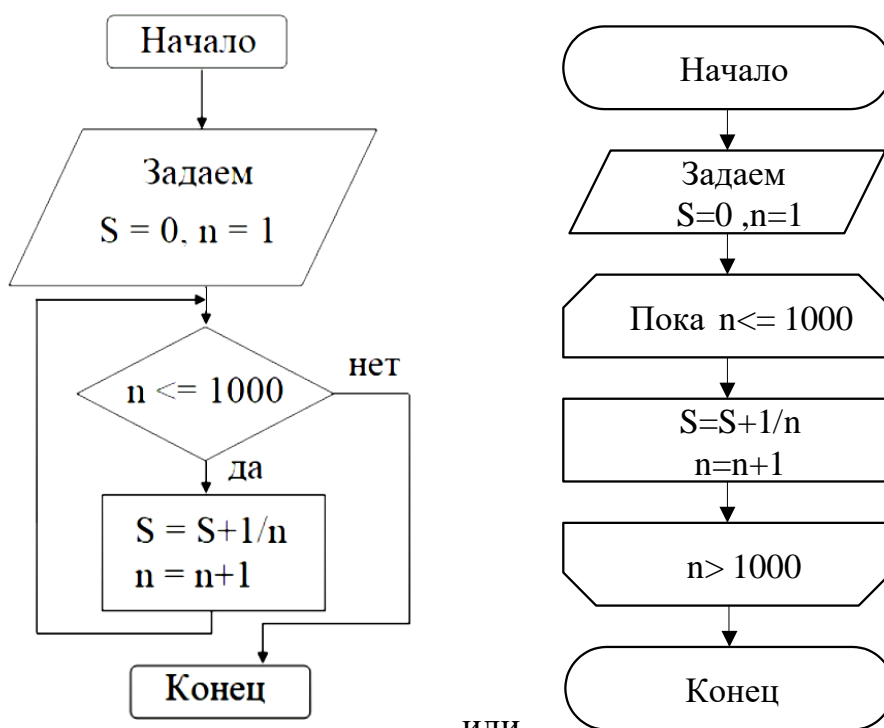


Рисунок 1.33 – Схема алгоритма поиска суммы чисел

Далее представлена программная реализация описанного алгоритма поиска суммы чисел:

```
#include <iostream>
using namespace std;
int main()
{ setlocale(LC_ALL, "rus");
    double S = 0;
    int n = 1;
    while (n <= 1000)
    {
        S += 1.0 / n;
        n++;
    }
    cout << S << endl;
    return 0;}
```

В данной программе сначала задается вещественная переменная *S*, которая будет хранить сумму ряда, затем счетчик *n*, изначальное значение которого 1. Далее идет цикл *while*, в круглых скобках записывается его условие (в данном случае $n \leq 1000$), и, пока оно истинно, выполняются два оператора: первый подсчитывает сумму ряда, а второй увеличивает значение счетчика на 1. Как только цикл *while* завершит свою работу, на экран выведется значение суммы. Результат выполнения данного программного кода представлен на рисунке 1.45.

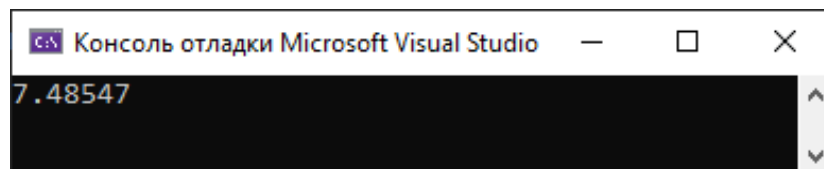


Рисунок 1.34 – Результат работы программы

1.4.4.3 Оператор цикла do while

Цикл do...while очень похож на цикл while. Единственное их различие в том, что при выполнении цикла do while один проход цикла будет выполнен независимо от условия.

Цикл do while имеет следующий синтаксис:

```
do  
{ [тело цикла] }  
while (<условие цикла>)
```

Далее приведен пример поиска суммы чисел от 1 до 1000 с применением цикла do...while:

```
#include <iostream>  
using namespace std;  
int main ()  
{  
    setlocale(0, "");  
    int i = 0; // инициализируем счетчик цикла  
    int sum = 0; // инициализируем счетчик суммы  
    do  
    {// выполняем цикл  
        i++;  
        sum += i; }  
    while (i < 1000); // пока выполняется условие  
    cout << "Сумма чисел от 1 до 1000 = " << sum << endl;  
    return 0;  
}
```

Реализация цикла do...while может использоваться, для проверки правильности ввода. Данная программа на языке C++ будет выглядеть следующим образом:

```
#include <iostream>  
using namespace std;
```

```

int main()
{
    setlocale(LC_ALL, "rus");
    const int secret_code = 13;
    int code_ent;
    do {        cout << "Введите секретный код: ";
                cin >> code_ent;
            } while (code_ent != secret_code);
    cout << "Вы ввели верный код!";
    return 0;
}

```

В данном примере сначала задается константа в виде целочисленной переменной, равной 13, данное число должен будет ввести пользователь. Затем выполняется тело цикла, т. е. пользователь вводит этот код. Далее идет проверка цикла, если введенный код не будет соответствовать секретному коду, то условие окажется истинным и цикл начнет работать заново. Если введенный код будет равен секретному коду, то произойдет выход из цикла.

Результат выполнения данного программного кода представлен на рисунке 1.35.

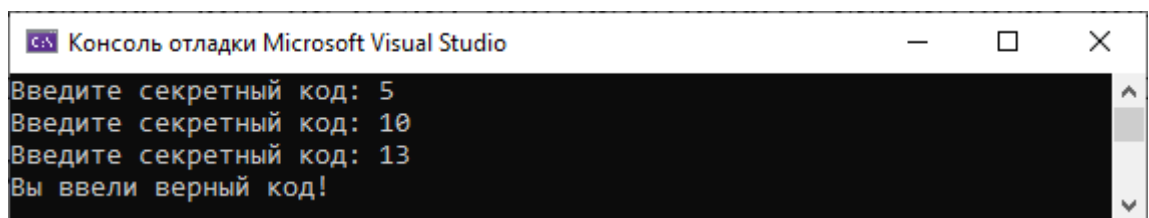


Рисунок 1.35 – Результат работы программы проверки ввода секретного кода

1.4.4.5 Операторы break и continue

Об операторе break уже есть упоминание выше, когда рассматривалась работа цикла for, с помощью оператора break прерывалась работа цикла for.

Также с помощью этого оператора прерывается бесконечный цикл. Когда его нужно прервать, в тело цикла добавляется условие, при выполнении которого сработает оператор break, а цикл завершит работу [1].

В следующем примере представлена работа с оператором `break`. Данная программа ищет сумму натуральных чисел до заданного числа `x`;

```
#include <iostream>

using namespace std;

int main() {

    setlocale(LC_ALL, "rus");

    int x, i, sum;

    sum = 0; //Начальное значение суммы
    x = 3; //Число до которого необходимо вычислять сумму
    i = 0; //Начальное значение текущего числа
    while (true) {

        if (i > x) { //Если текущее число больше заданного
            break; //Прерываем цикл
        }

        sum += i; //Вычисляем сумму
        i++; //Переходим к следующему числу
    }

    cout << sum; //Выводим значение суммы
    return 0;
}
```

Результат данной программы для числа 3 представлен на рисунке 1.36



Рисунок 1.36 – Результат работы оператора `break`

Оператор `continue` пропускает тело цикла, которое стоит после него. Этот оператор применяется тогда, когда необходимо прервать (пропустить) текущую итерацию цикла и приступить к следующей итерации. При его выполнении в

цикле for происходит остановка текущей итерации, переход к изменению управляющей переменной, а затем проверка условия продолжения выполнения цикла.

Далее приведен пример использования оператора continue для решения следующей задачи. Необходимо вывести на экран те числа, которые делятся на 7 без остатка и находятся в диапазоне от 1 до 70:

```
#include <iostream>
using namespace std;
int main(){
    for (int i = 1; i <=70; i++){
        if (i % 7 != 0) // если число не делится на 7 без остатка
        { continue; // прервать эту итерацию цикла и выполнить i++
        }
        cout << i << endl; }
    return 0; }
```

Результат выполнения данного программного кода представлен на рисунке 1.37

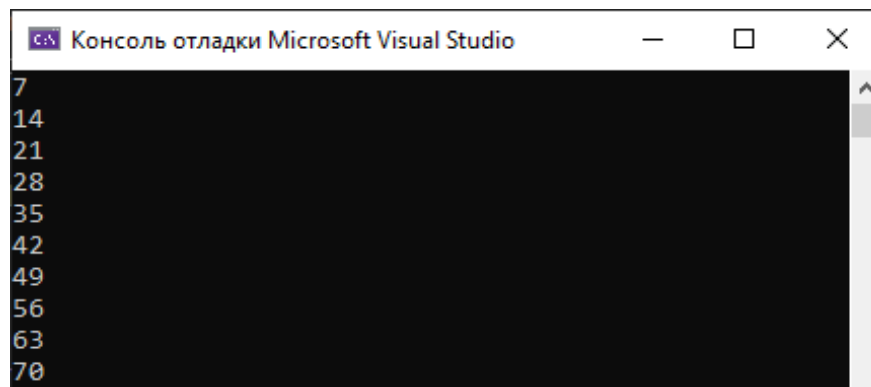


Рисунок 1.37 – Результат работы оператора continue

Если число i не делится на 7 без остатка (остаток от деления не равен 0), сработает continue. То есть строка кода 12 уже не выполнится, и мы не увидим число i на экране. В случае если остаток от деления i на 7 будет равен 0 (число делится на 7 нацело), тело if выполнено не будет, и число будет показано.