Ravesli Ravesli

- Уроки по C++
- OpenGL
- SFML
- Qt5
- RegExp
- Ассемблер
- Купить .PDF

# Урок №72. Обработка некорректного пользовательского ввода

Юрий |

- Уроки C++

|

Обновл. 24 Сен 2020 |

63006

| 15

Большинство программ, имеющих хоть какой-либо пользовательский интерфейс, сталкиваются с обработкой пользовательского ввода. В программах, которые мы писали раньше, используется **std::cin** для получения пользовательского ввода. А так как ввод данных имеет свободную форму (пользователь может ввести всё, что пожелает), то пользователю очень легко ввести данные, которые от него никак не ожидаются.

При написании программ, вы всегда должны учитывать, как пользователи (преднамеренно или непреднамеренно) могут использовать ваши программы. Хорошо написанная программа предвидит, как ею могут злоупотребить, и либо будет обрабатывать такие случаи, либо предотвращать их (если это возможно). Программа, которая имеет обработку некорректного ввода, называется **надежной**.

На этом уроке мы рассмотрим, как пользователи могут вводить некорректные данные через std::cin, а также как с этим бороться.

Оглавление:

# std::cin, буфер данных и извлечение

Прежде чем разбираться с обработкой некорректного ввода через std::cin и оператор >>, давайте сначала рассмотрим их принцип работы.

Процесс, когда мы используем оператор >> для получения данных от пользователя и размещение этих данных в определенной переменной, называется **извлечением**. Соответственно, **оператор >>** называется **оператором извлечения**.

Когда пользователь вводит данные в ответ на операцию извлечения, то эти данные помещаются в буфер std::cin. **Буфер данных** — это просто часть памяти, зарезервированная для временного хранения данных, когда они перемещаются из одного места в другое. В этом случае буфер используется для хранения пользовательского ввода, пока он находится в режиме ожидания выделения для него переменных.

При использовании оператора извлечения, выполняется следующая процедура:

➔ Если во входном буфере есть данные, то эти данные используются для извлечения.

➔ Если во входном буфере нет данных, то пользователю предлагается ввести данные (обычно именно это и происходит в большинстве случаев). Когда пользователь нажимает Enter, символ новой строки \n помещается во входной буфер.

➔ Оператор >> извлекает столько данных из входного буфера в переменную, сколько позволяет **размер** самой переменной (игнорируя любые пробелы, табы или \n).

➔ Любые данные, которые не были извлечены, остаются во входном буфере для последующего извлечения.

Извлечение данных считается успешным, если по крайней мере один символ был извлечен из входного буфера. Оставшиеся данные во входном буфере используются для последующих извлечений. Например:

```
1 int a;
2 std::cin >> a;
```

Если пользователь введет 7d, то 7 будет извлечено, преобразовано в целое число и присвоено переменной a. А d\n останется во входном буфере дожидаться следующего извлечения.

Извлечение не выполняется, если данные ввода не соответствуют типу переменной, выделенной для них. Например:

```
1 int a;
2 std::cin >> a;
```

Если бы пользователь ввел z, то извлечение не выполнилось бы, так как z не может быть извлечено в целочисленную переменную.

# Проверка пользовательского ввода

Существует три основных способа проверки пользовательского ввода:

- До ввода

  → Предотвращение некорректного пользовательского ввода.

- После ввода

  → Пользователь может вводить всё, что хочет, но осуществляется последующая проверка данных. Если проверка прошла успешно, то выполняется перемещение данных в переменную.

  → Пользователь может вводить всё, что хочет, но при операции извлечения данных оператором >> параллельно решаются возможные ошибки.

Некоторые графические пользовательские интерфейсы или расширенные текстовые интерфейсы позволяют проверять ввод пользователя сразу (символ за символом). Программист создает функцию проверки данных, которая принимает и проверяет пользовательский ввод, и, если данные ввода корректны, то возвращается true, если нет — false. Эта функция вызывается каждый раз, когда пользователь нажимает на клавишу. Если функция проверки возвращает true, то символ, который пользователь ввел — принимается. Если функция возвращает false, то символ, который только что ввел пользователь — отбрасывается (и не отображается на экране). Используя этот метод, мы можем гарантировать, что любой пользовательский ввод будет корректным, так как любые неверные нажатия клавиш будут обнаружены и немедленно удалены. К сожалению, std::cin не поддерживает этот тип проверки.

Поскольку **строки** не имеют никаких ограничений на то, какие символы вводятся, то извлечение гарантированно будет успешным (хотя помним, что std::cin останавливает процесс извлечения при первом обнаружении символа пробела). После ввода строки программа сразу может её проанализировать. Однако этот анализ и последующая конвертация данных в другие типы данных (например, числа) может быть сложной, поэтому это делается только в редких случаях.

Чаще всего мы позволяем std::cin и оператору извлечения выполнять тяжелую работу. В соответствии с этим методом пользователь может вводить всё, что хочет, а далее std::cin и оператор >> пытаются извлечь данные и, если что-то пойдет не так, выполняется **обработка возможных ошибок**. Это самый простой способ, и его мы и будем рассматривать.

Рассмотрим следующую программу «Калькулятор», которая не имеет обработки ошибок:

```
#include <iostream>

double getValue()
{
    std::cout << "Enter a double value: ";
    double a;
    std::cin >> a;
    return a;
}

char getOperator()
```

```
{
    std::cout << "Enter one of the following: +, -, *, or /: ";
    char sm;
    std::cin >> sm;
    return sm;
}

void printResult(double a, char sm, double b)
{
    if (sm == '+')
        std::cout << a << " + " << b << " is " << a + b << '\n';
    else if (sm == '-')
        std::cout << a << " - " << b << " is " << a - b << '\n';
    else if (sm == '*')
        std::cout << a << " * " << b << " is " << a * b << '\n';
    else if (sm == '/')
        std::cout << a << " / " << b << " is " << a / b << '\n';
}

int main()
{
    double a = getValue();
    char sm = getOperator();
    double b = getValue();

    printResult(a, sm, b);

    return 0;
}
```

Здесь мы просим пользователя ввести два числа и арифметический оператор. Результат выполнения программы:

```
Enter a double value: 4
Enter one of the following: +, -, *, or /: *
Enter a double value: 5
4 * 5 is 20
```

Теперь рассмотрим, где некорректный ввод пользователя может привести к сбою в программе.

Сначала мы просим пользователя ввести первое число. А что будет, если он введет что-либо другое (например, `q`)? В этом случае извлечение данных не произойдет.

Во-вторых, мы просим пользователя ввести один из 4-х возможных символов (**арифметических операторов**). Что будет, если он введет какой-то другой символ? Мы сможем извлечь данные, но не сможем их обработать.

В-третьих, что будет, если пользователь вместо символа введет строку, например, `*q hello`. Хотя мы можем извлечь символ *, но в буфере останется лишний балласт, который в будущем может привести к проблемам.

# Основные типы некорректного пользовательского ввода

Мы выделили 4 типа:

- Извлечение выполняется успешно, но значения бесполезны для программы (например, вместо математического оператора введен символ `k`).
- Извлечение выполняется успешно, но пользователь вводит лишний текст (например, `*q hello` вместо одного символа математического оператора).
- Извлечение не выполняется (например, вместо числового значения ввели символ `q`).
- Извлечение выполняется успешно, но пользователь ввел слишком большое числовое значение.

Таким образом, чтобы наши программы были надежными, то всякий раз, когда мы просим пользователя ввести данные, мы должны предугадать возможность возникновения любого из вышеуказанных событий, и, если это произойдет, в программе должна быть обработка таких случаев.

Давайте рассмотрим каждую из ситуаций, приведенных выше, а также алгоритм действий, если такая ситуация случилась.

# Ошибка №1: Извлечение выполняется успешно, но данные бесполезны

Это самый простой случай. Рассмотрим следующий фрагмент выполнения программы, приведенной выше:

```
Enter a double value: 4
Enter one of the following: +, -, *, or /: k
Enter a double value: 5
```

Здесь мы просим пользователя ввести один из 4-х арифметических операторов, но он вводит `k`. Символ `k` является допустимым символом, поэтому std::cin извлекает его в переменную `sm`, и всё это добро возвращается обратно в функцию main(). Но в программе не предусмотрен случай обработки подобного ввода, поэтому мы получаем сбой, и в результате ничего не выводится.

Решение простое: выполнить проверку пользовательского ввода. Обычно она состоит из 3-х шагов:

- Проверить пользовательский ввод на ожидаемые значения.
- Если ввод совпадает с ожидаемым, то значения благополучно возвращаются.
- Если нет, то сообщаем пользователю что что-то пошло не так, и просим повторить ввод снова.

Вот обновленная функция getOperator() с проверкой пользовательского ввода:

```
1 char getOperator()
2 {
```

```
    while (true) // цикл продолжается до тех пор, пока пользователь не введет корректно
    {
        std::cout << "Enter one of the following: +, -, *, or /: ";
        char sm;
        std::cin >> sm;

        // Выполняем проверку значений
        if (sm == '+' || sm == '-' || sm == '*' || sm == '/')
            return sm; // возвращаем данные в функцию main()
        else // в противном случае, сообщаем пользователю, что что-то пошло не так
            std::cout << "Oops, that input is invalid.  Please try again.\n";
        }
}
```

Мы используем **цикл while** для гарантии корректного ввода. Если пользователь введет один из ожидаемых символов, то всё хорошо — символ возвратится обратно в функцию main(), если нет — пользователю выведется просьба повторить ввод снова. И вводить данные он будет до тех пор, пока не введет корректное значение, не закроет программу или не уничтожит свой компьютер 🙂

## Ошибка №2: Извлечение выполняется успешно, но пользователь вводит лишний текст

Рассмотрим следующее выполнение программы «Калькулятор»:

```
Enter a double value: 4*5
```

Как вы думаете, что произойдет дальше?

```
Enter a double value: 4*5
Enter one of the following: +, -, *, or /: Enter a double value: 4 * 5 is 20
```

Программа выведет верный результат, но её порядок выполнения неправильный. Почему? Сейчас разберемся.

Когда пользователь вводит `4*5`, то эти данные поступают в буфер. Затем оператор `>>` извлекает `4` в переменную `a`, оставляя `*5\n` в буфере. Затем программа выводит `Enter one of the following: +, -, *, or /:`. Однако, когда вызывается оператор извлечения, он видит, что в буфере находится `*5\n`, поэтому он использует эти данные вместо того, чтобы запрашивать их у пользователя еще раз. Следовательно, извлекается символ *, а `5\n` остается во входном буфере.

После того, как пользователя просят ввести другое число, `5` извлекается из буфера, не дожидаясь ответа от пользователя. Поскольку у пользователя не было возможности ввести другие значения и нажать Enter (вставляя символ новой строки), то всё происходит на одной строке.

Хотя результат программы правильный, но её выполнение — нет. Согласитесь, что с наличием возможности просто проигнорировать лишние символы было бы намного лучше. К счастью, это можно сделать следующим образом:

```
std::cin.ignore(32767, '\n'); // удаляем до 32767 символов из входного буфера вплоть до
```

Поскольку последний символ, введенный пользователем, должен быть `\n` (так как пользователь в конце ввода нажимает Enter), то мы можем сообщить std::cin игнорировать все символы в буфере до тех пор, пока не будет найден символ новой строки (который мы также удаляем). Таким образом, всё лишнее будет успешно проигнорировано.

Обновим нашу функцию getDouble(), добавив эту строчку:

```
double getValue()
{
    std::cout << "Enter a double value: ";
    double a;
    std::cin >> a;
    std::cin.ignore(32767, '\n'); // удаляем до 32767 символов из входного буфера вплоть
    return a;
}
```

Теперь наша программа будет работать так, как надо, даже если мы введем `4*5` в первой строке. Число `4` будет извлечено в переменную, а все остальные символы будут удалены из входного буфера. Поскольку входной буфер теперь пуст, то при последующем выполнении операции извлечения всё пройдет гладко и порядок выполнения программы не будет нарушен.

# Ошибка №3: Извлечение не выполняется

Рассмотрим следующее выполнение программы «Калькулятор»:

```
Enter a double value: a
```

Теперь уже не удивительно, что программа работает не так, как надо, но её дальнейший ход выполнения — вот что интересно:

```
Enter a double value: a
Enter one of the following: +, -, *, or /: Enter a double value:
```

И программа внезапно обрывается.

Этот случай похож на ошибку №2, но все же несколько отличается. Давайте рассмотрим детально, что здесь происходит.

Когда пользователь вводит `a`, то этот символ помещается в буфер. Затем оператор `>>` пытается извлечь `a` в переменную `a` типа double. Поскольку `a` нельзя конвертировать в **тип double**, то оператор `>>` не может выполнить извлечение. На этом этапе случаются две вещи: `a` остается в буфере, а std::cin переходит в «**режим отказа**». Как только установлен этот режим, то все последующие запросы на извлечение данных будут проигнорированы.

К счастью, мы можем определить, удачно ли прошло извлечение или нет. Если нет, то мы можем исправить ситуацию следующим образом:

```
if (std::cin.fail()) // если предыдущее извлечение было неудачным,
```

```
{
    std::cin.clear(); // то возвращаем cin в 'обычный' режим работы
    std::cin.ignore(32767,'\n'); // и удаляем значения предыдущего ввода из входного бу
}
```

Вот так! Теперь давайте интегрируем это в нашу функцию getValue():

```
double getValue()
{
    while (true) // цикл продолжается до тех пор, пока пользователь не введет корректн
    {
        std::cout << "Enter a double value: ";
        double a;
        std::cin >> a;

        if (std::cin.fail()) // если предыдущее извлечение оказалось неудачным,
        {
            std::cin.clear(); // то возвращаем cin в 'обычный' режим работы
            std::cin.ignore(32767,'\n'); // и удаляем значения предыдущего ввода из вх
        }
        else // если всё хорошо, то возвращаем a
            return a;
    }
}
```

## Ошибка №4: Извлечение выполняется успешно, но пользователь ввел слишком большое числовое значение

Рассмотрим следующий код:

```
#include <iostream>
#include <cstdint>

int main()
{
    std::int16_t x { 0 }; // переменная x занимает 16 бит, её диапазон значений: от -3
    std::cout << "Enter a number between -32768 and 32767: ";
    std::cin >> x;

    std::int16_t y { 0 }; // переменная y занимает 16 бит, её диапазон значений: от -3
    std::cout << "Enter another number between -32768 and 32767: ";
    std::cin >> y;

    std::cout << "The sum is: " << x + y << '\n';
    return 0;
}
```

Что случится, если пользователь введет слишком большое число (например, `40000`)?

```
Enter a number between -32768 and 32767: 40000
Enter another number between -32768 and 32767: The sum is: 32767
```

В вышеприведенном примере std::cin немедленно перейдет в «режим отказа», и значение не будет присвоено переменной `x`. Следовательно, переменной `x` присваивается максимально возможное значение типа данных (в этом случае, `32767`). Все следующие данные ввода будут проигнорированы, а `y` останется с инициализированным значением `0`. Решение такое же, как и в случае с неудачным извлечением (см. ошибка №3).

## Объединяем всё вместе

Вот программа «Калькулятор», но уже с полным механизмом обработки/проверки ошибок:

```
#include <iostream>

double getValue()
{
    while (true) // цикл продолжается до тех пор, пока пользователь не введет корректно
    {
        std::cout << "Enter a double value: ";
        double a;
        std::cin >> a;

        // Проверка на предыдущее извлечение
        if (std::cin.fail()) // если предыдущее извлечение оказалось неудачным,
        {
            std::cin.clear(); // то возвращаем cin в 'обычный' режим работы
            std::cin.ignore(32767,'\n'); // и удаляем значения предыдущего ввода из вх
            std::cout << "Oops, that input is invalid.  Please try again.\n";
        }
        else
        {
            std::cin.ignore(32767,'\n'); // удаляем лишние значения

            return a;
        }
    }
}

char getOperator()
{
    while (true) // цикл продолжается до тех пор, пока пользователь не введет корректно
    {
        std::cout << "Enter one of the following: +, -, *, or /: ";
        char sm;
```

```
        std::cin >> sm;

        // Переменные типа char могут принимать любые символы из пользовательского вво

        std::cin.ignore(32767,'\n'); // удаляем лишний балласт

        // Выполняем проверку пользовательского ввода
        if (sm == '+' || sm == '-' || sm == '*' || sm == '/')
            return sm; // возвращаем обратно в caller
        else // в противном случае, сообщаем пользователю что что-то пошло не так
            std::cout << "Oops, that input is invalid.  Please try again.\n";
        }
}

void printResult(double a, char sm, double b)
{
    if (sm == '+')
        std::cout << a << " + " << b << " is " << a + b << '\n';
    else if (sm == '-')
        std::cout << a << " - " << b << " is " << a - b << '\n';
    else if (sm == '*')
        std::cout << a << " * " << b << " is " << a * b << '\n';
    else if (sm == '/')
        std::cout << a << " / " << b << " is " << a / b << '\n';
    else
        std::cout << "Something went wrong: printResult() got an invalid operator.";

}

int main()
{
    double a = getValue();
    char sm = getOperator();
    double b = getValue();

    printResult(a, sm, b);

    return 0;
}
```

# Заключение

При написании программ, всегда думайте о том, как пользователи могут злоупотребить ими или использовать не по назначению, особенно то, что касается ввода данных. Подумайте:

- Может ли извлечение не выполниться?
- Может ли пользователь ввести значение больше ожидаемого?

- Может ли пользователь ввести бессмысленные значения?
- Может ли ввод пользователя привести к **переполнению** переменных?

Используйте **ветвление if** и **логические переменные** для проверки пользовательского ввода.

Следующий код осуществляет проверку пользовательского ввода и исправляет проблемы с переполнением или неудачным извлечением данных:

```
if (std::cin.fail()) // если предыдущее извлечение не выполнилось или произошло перепол
{
    std::cin.clear(); // то возвращаем cin в 'обычный' режим работы
    std::cin.ignore(32767,'\n'); // и удаляем значения предыдущего ввода из входного бу
}
```

***Примечание:*** Проверка пользовательского ввода очень важна и полезна, но она, к сожалению, приводит к усложнению кода, что, в свою очередь, затрудняет его понимание. Поэтому на следующих уроках мы не будем её использовать, дабы всё оставалось максимально простым и доступным.

Оценить статью:

(**286** оценок, среднее: **4,93** из 5)

Урок №71. Генерация случайных чисел

Урок №73. Введение в тестирование кода

# Комментариев: 15

1. *vik*:
   19 августа 2019 в 16:19

   "std::int16_t x { 0 }; // переменная x занимает 16 бит, её диапазон значений: от -32 768 до 32 767"
   …
   "Следовательно, x останется с инициализированным значением 32767"

   Почему 32767, если переменная инициализирована значением 0?

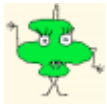
   Ответить

   1. *Жека*:
      26 ноября 2020 в 15:36

Наверно, вот об этом "Следовательно, переменной x присваивается максимально возможное значение типа данных в этом случае, 32767"

Тоже бы послушал почему не 0.

Ответить

2.  *Алексей*:
[23 июля 2019 в 13:29](#)

В случаи один лучше использовать RegEX, но это другая история)

Ответить

3.  *zvezdonom*:
[14 апреля 2019 в 00:15](#)

Вот переделал функцию getValue().
Теперь если ввести что-то такое "4657лапивпо", то уже не
прокатит. Это ответ Alexey-ю. Так же не прокатит такой ввод "влпр 474".

```
#include <iostream>
#include <string>

double getValue()
{
    while (true) // Цикл продолжается до тех пор, пока пользователь не введет кор
    {
        std::cout << "Enter a double value: ";

        double a;
        bool flag = true;
        std::string str;
        std::cin >> str;

        for(int i = 0; i < str.size(); i++)
        {
            if(str.at(i) != '0' && str.at(i) != '1' && str.at(i) != '2' && str.at
                    str.at(i) != '5' && str.at(i) != '6' && str.at(i) != '7' && s
            {
                flag = false;
                std::cout << "Oops, that input is invalid.  Please try again.\n";
                std::cin.ignore(32767,'\n'); // удаляем лишние значения
                break;
            }
        }
        if(flag)
        {
```

```
            a = atof(str.c_str()); // atof преобразует строку в значение типа dou
            std::cin.ignore(32767,'\n'); // удаляем лишние значения
            return a;
        }
    }
}

char getOperator()
{
    while (true) // Цикл продолжается до тех пор, пока пользователь не введет кор
    {
        std::cout << "Enter one of the following: +, -, *, or /: ";
        char sm;
        std::cin >> sm;

        // Переменные типа char могут принимать любые символы из пользовательског

        std::cin.ignore(32767,'\n'); // удаляем лишний балласт

        // Проверяем пользовательские значения на корректность
        if (sm == '+' || sm == '-' || sm == '*' || sm == '/')
            return sm; // возвращаем в caller
        else // в противном случае сообщаем пользователю что что-то пошло не так
            std::cout << "Oops, that input is invalid.  Please try again.\n";
    }
}

void printResult(double a, char sm, double b)
{
    if (sm == '+')
        std::cout << a << " + " << b << " is " << a + b << '\n';
    else if (sm == '-')
        std::cout << a << " - " << b << " is " << a - b << '\n';
    else if (sm == '*')
        std::cout << a << " * " << b << " is " << a * b << '\n';
    else if (sm == '/')
        std::cout << a << " / " << b << " is " << a / b << '\n';
    else
        std::cout << "Something went wrong: printResult() got an invalid operator

}

int main()
{
    double a = getValue();
    char sm = getOperator();
    double b = getValue();

```

```
77      printResult(a, sm, b);
78
79      return 0;
    }
```

Ответить

1.  *Andrey*:
   13 сентября 2019 в 18:08

   Пропущена библиотека #include <cstdlib>, не забудьте подключить

   Ответить

4.  *Alex*:
   9 февраля 2019 в 23:10

   В конце статьи конечная версия кода для проверки вводимого значения, как я понял. Но я нашел уязвимое место. Да, он подходит, если мы вводим только буквы (gsrjtykj), буквы-цифры (fawy426), но если ввести цифры-буквы (15gdat), то код становится бесполезным.

   Ответить

   1.  *Molex*:
      27 февраля 2019 в 12:20

      Ну так предполагается, что ты будешь сбрасывать балласт из букв.

      Ответить

5.  *Alexey*:
   8 февраля 2019 в 13:45

   Я вот не понял, если мне надо ввести число int, пользователь ввел, например 4545врвр, то используя if (std::cin.fail()) компилятор присвоит 4545 переменной, а "врвр" отбросит. А как ему сообщит что это ввод полностью неверный?

   Ответить

6.  *Алексей*:
   3 ноября 2018 в 13:55

   Подскажите пожалуйста, а как программа определяет что предыдущий ввод провальный?

```
1 if (std::cin.fail()) // если предыдущее извлечение было провальным
```

   Ответить

1.  *Евгений*:
   10 марта 2019 в 05:13

   cin.fail() даст true если ввод был некорректный т.е cin вошел в режим отказа.

   Ответить

7.  *Герман*:
   31 января 2018 в 18:21

   Понял. Спасибо!!!

   Ответить

   1. *Юрий*:
      31 января 2018 в 18:30

      Пожалуйста 🙂

      Ответить

8.  *Герман*:
   31 января 2018 в 17:03

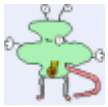
   Уважаемый автор, поясните, пожалуйста, как появляется цифра 32767 в коде (в двоичной системе исчисления это 15 единиц), переменная double — 8 байтов, это 1.844674407371E+19.

   Ответить

   1. *Юрий*:
      31 января 2018 в 18:03

      cin.ignore — это стандартное решение из библиотеки iostream. Число, которое указывается в скобках — выборочное, можно указать 1, можно 100, можно 270, а можно 32767. Это число не является результатом каких-либо вычислений, вы указываете то, что хотите сами — сколько символов вы считаете нужным убрать из входного буфера.

      Ответить

      1.  *Юлия*:
         14 октября 2019 в 23:18

         32767 — это максимум, который помещается в int. Т.е. для int игнорировать 32767 то же самое, что игнорировать "всё". Правильно?

Ответить

## Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены *

Имя *

Email *

Комментарий

Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию

Получать уведомления о новых комментариях по электронной почте. Вы можете подписаться без комментирования.

Отправить комментарий

TELEGRAM КАНАЛ

ПАБЛИК

ТОП СТАТЬИ

- Словарь программиста. Сленг, который должен знать каждый кодер
- Урок №1. Введение в программирование
- 70+ бесплатных ресурсов для изучения программирования
- Урок №1: Введение в создание игры «SameGame» на C++/MFC
- Урок №4. Установка IDE (Интегрированной Среды Разработки)