УНИВЕРСИТЕТ СИНЕРГИЯ

РАЗРАБОТКА НА C++

Урок 8.
Обсуждение условных и циклических конструкций, закрепление знаний о строках и массивах

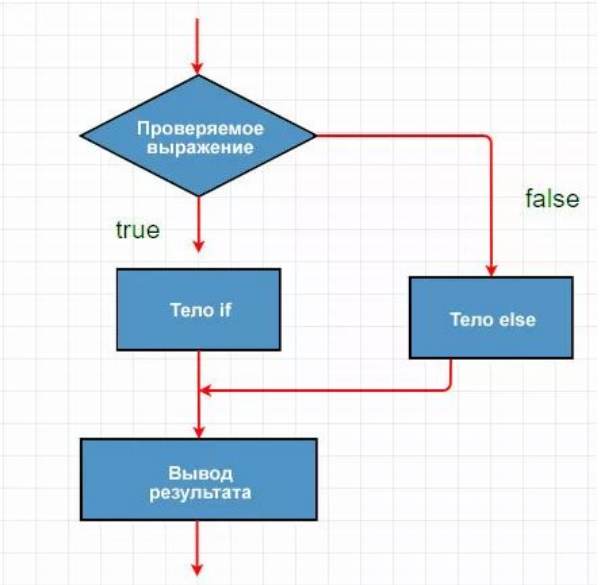
Обсуждение темы 3
«Булевый тип данных и условия»

Структура условной конструкции

Условная конструкция в C++ состоит из двух частей: ветки «if» и ветки «else». После слова «if» описывается проверяемое условие, далее следуют команды, которые выполняются, если условие истинно.
Ветка «else» идет следом за веткой «if». В ветке «else» размещают команды, которые выполняются если проверяемое условие всё-таки ложно.

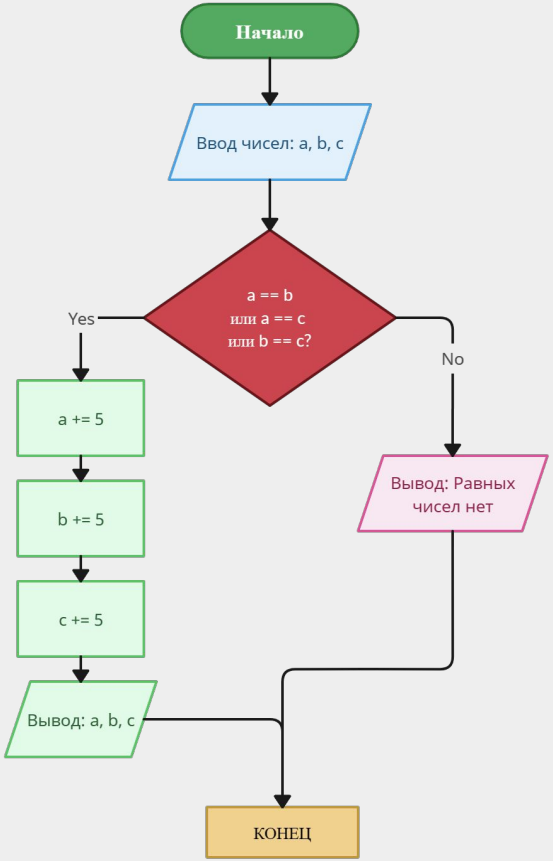
Синтаксис условной конструкции:

```
if (условие) {
    // код выполняется, если условие истинно
} else {
    // код выполняется, если условие ложно
}
```



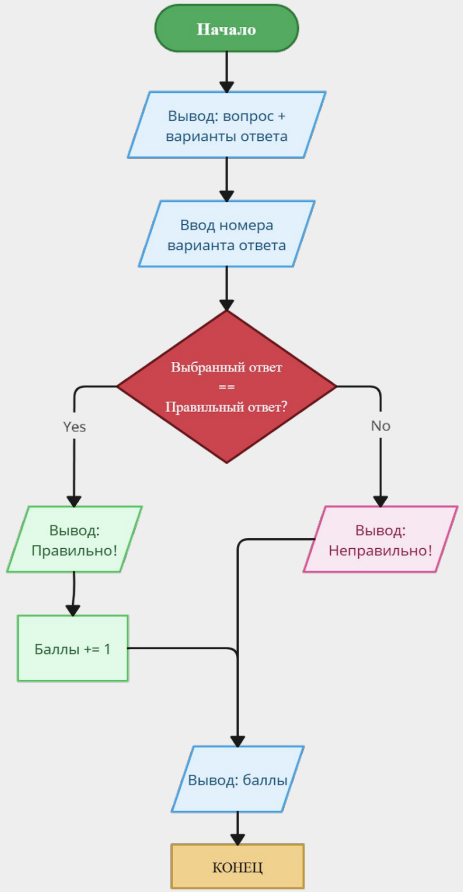
Разбор 3 ДЗ

Блок-схема к задаче № 1



Разбор 3 ДЗ

Блок-схема к задаче № 2



Обсуждение темы 4
«Цикл “while”»

«while» — повторяющийся «if»

Инструкции внутри if выполняются только один раз, и это происходит только если условие в if истинно (true). Если условие ложно (false), инструкции внутри if пропускаются.

Пример:

```
int x = 5;
if (x > 0) {
    std::cout << "Это выполнится 1 раз" << std::endl;
}
```

«while» — повторяющийся «if»

Цикл «while» выполняет инструкции внутри себя многократно до тех пор, пока условие, заданное после ключевого слова while, остается истинным, что делает его похожим на "повторяющийся if". Количество выполнений инструкций внутри цикла зависит от того, как долго условие остается истинным.
Пример:

```
int count = 5;
while (count > 0) {
        std::cout << "Это выполняется " << count << " раз." << std::endl;
        count--; // Уменьшаем счетчик на 1 при каждой итерации
}
```

В этом примере цикл while выполняет инструкции внутри него 5 раз, потому что count начинается с 5 и уменьшается на каждой итерации до 0.

Разбор домашнего задания по теме «Цикл “while”»

Обсуждение темы 5
«Цикл “for”»

Синтаксис цикла «for»

Цикл for имеет следующий общий синтаксис:

```
for (инициализация; условие; выражение обновления) {
    // Тело цикла
}
```

Оператор "инициализация" выполняется один раз в начале цикла и используется для инициализации переменных, которые будут участвовать в цикле.

Условие задает условие продолжения выполнения цикла. Пока условие истинно, цикл будет выполняться.

"Выражение обновления" выполняется после каждой итерации цикла и обычно используется для изменения переменных контроля цикла.

Параметры цикла «for»: инициализация

Инициализация - это объявление переменной, которая будет использоваться в цикле.

Например:

```
for (int i = 0; i < 5; i++) {
    // Тело цикла
}
```

Параметры цикла «for»: условие

Условие определяет, когда цикл будет выполняться и когда он завершится. Цикл будет выполняться, пока это условие истинно.

Например:

```
for (int i = 0; i < 5; i++) {
        // Тело цикла будет выполняться пять раз (i = 0, 1, 2, 3, 4)
}
```

Параметры цикла «for»: выражение обновления

Это выражение, которое выполняется после каждой итерации цикла. Обычно оно используется для изменения переменных цикла.

Например:

```
for (int i = 0; i < 5; i++) {
    // После каждой итерации i увеличится на 1
}
```

Разбор домашнего задания по теме «Цикл “for”»

Обсуждение темы 6
«Строки»

Методы строк

Метод size(). Используется для того, чтобы узнать количество символов в строке. Возвращает целое число. Запись: s1.size();

Метод empty(). Проверяет пустая наша строка или нет. Если пустая, то возвращает true. В ином случае - false. Запись: s1.empty();

Метод erase(). Используется для того, чтобы удалить часть строки.

- s3.erase(2); // удаляется всё после 2 символа
- s6.erase(2, 4); // удаляется всё от 2 до 4 символа

Метод insert(). Используется для того, чтобы вставить некоторые символы в строку.

- s2.insert(1, 3, '!'); // вставляем 3 восклицательных знака, начиная от позиции 1
- s3.insert(4, s2); // вставляем то, что у строке s2 в строку s3, начиная от позиции 4
- s5.insert(0, s3, 3, 2); // вставляем в строку s5 два символа из строки s3 от символа 3 строки s3. Начальная позиция вставки - 0

Методы строк

Метод find(). Используется, чтобы найти от какого индекса в строке начинается, нужная последовательность символов. Возвращает целое число, равное индексу.

- s5.find(s4); //ищем, где в первый раз появляется строка s4 полностью в строке s5. Поиск начинаем от индекса 0.
- s6.find("d", 4); // ищем, где в первый раз появляется символ d в строке s6. Поиск начинаем от индекса 4.
- s3.find(s2, 2, 3); //ищем первые 3 символа строки s2 в строке s3, начиная от индекса 2.

Метод getline(). Используется, чтобы получить предложение (строку с пробелами).
Пример:

```
string s7; //объявляем две строки
string s8;
cin >> s7; //вводим s7 через cin. Если поставить пробел, то выведутся только символы до пробела
getline(cin, s8); //вводим s8 через поток cin с помощью getline. Если поставить пробел, то всё равно введутся все символы.
```

Разбор домашнего задания по теме «Строки»

Обсуждение темы 7
«Массивы и ГПСЧ»

Понятие массива

- **Массив** – упорядоченная последовательность однотипных данных, объединенных под одним именем

- **Mas[0], Mas[1]** – обращение к первому и второму элементу массива

- Доступ к отдельному элементу массива осуществляется с помощью **индекса**
- В C++ все массивы состоят из **соприкасающихся участков памяти.** Наименьший адрес соответствует первому элементу, наибольший адрес – последнему элементу
- Каждый элемент массива может использоваться **как отдельная переменная**
- Массивы могут быть **многомерными**

Инициализация массива

- Объявлен массив типа int размером 5 и **полностью инициализирован**

```
int array1[5] = {1,2,3,4,5};
```

- Объявлен массив типа char размером 5, но инициализировано **только 3** элемента (главное, что **не больше 5**)

```
char array1[5] = {'a', 'b', 'c'};
```

- Объявлен массив типа int, размер которого компилятор определяет **при анализе инициализатора**

```
int array1[] = {1,2,3};
```

- Объявлен массив типа int размером 65

```
int array1['A'];
```

Разбор домашнего задания по теме «Массивы и ГПСЧ»

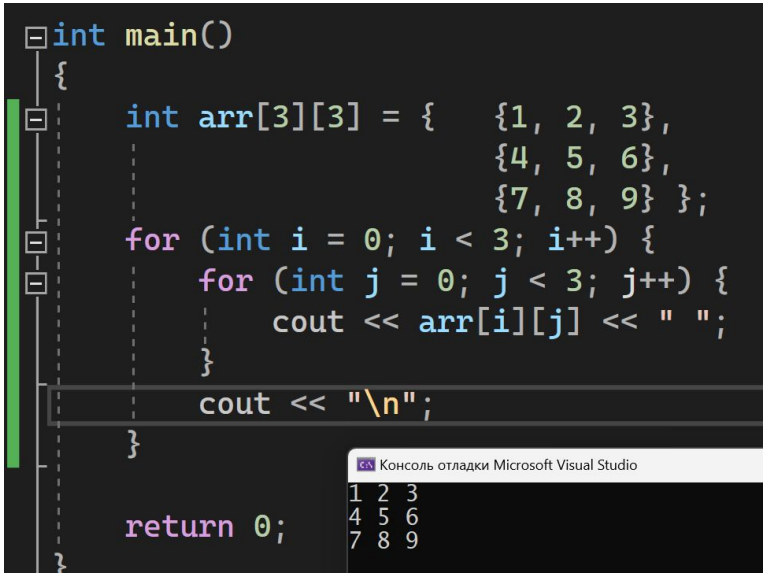
Двумерные массивы

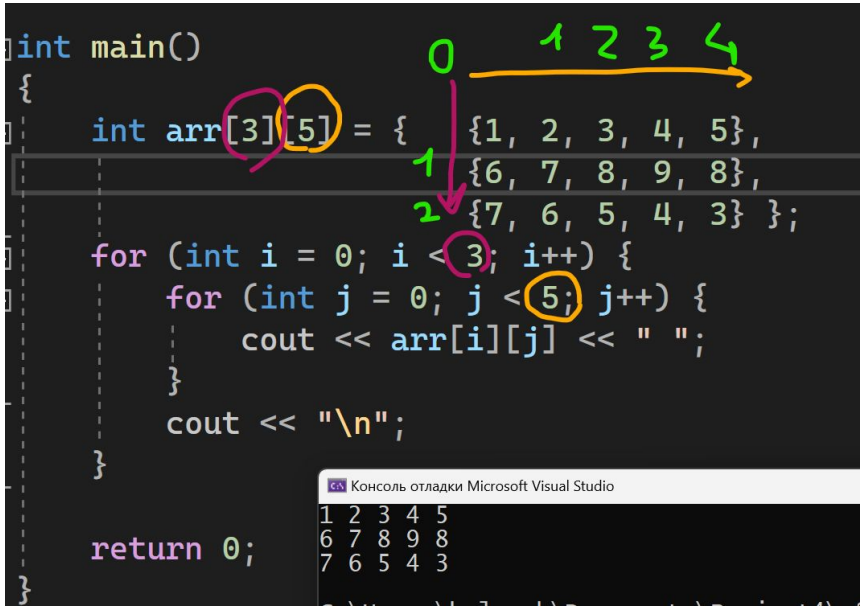
Обычный массив выглядит так:
int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
и хранится в памяти это соответственно arr[0] -> 0, arr[1] -> 1… и т.д.
Давайте попробуем создать массив с двумя размерами:
int arr[3][3] = {{1, 2, 3},{4, 5, 6},{7, 8, 9}};
Выглядит сложно :) Добавим пробелы и увидим таблицу (строки и столбцы)

```
int arr[3][3] = {       {1, 2, 3},
                        {4, 5, 6},
                        {7, 8, 9} };
```

Мы можем хранить внутри одного массива, сколько угодно других массивов, при этом первые [] будут отвечать за индекс нужного массива (нужную строку), а вторые [] за конкретный элемент этого массива:
arr[0] -> {1, 2, 3};
arr[0][0] -> 1,
arr[0][1] -> 2,
….и т.д.,
arr[2][2] -> 9

Двумерные массивы





Динамические массивы

```
int arr[10];
int arr[11]; //Ошибка!

int *nums = new int[10];
nums = new int[11]; //ОК, но старый массив будет потерян
```

* - позволяет создать указатель на область в памяти
new - специальное слово для выделения памяти

Важно! При создании нового массива в одном и том же указателе - старый массив остается в памяти до конца работы программы, просто **мы теряем единственную переменную, которая хранила его адрес** и записываем туда новый адрес.

Динамические массивы

Это может привести к утечке памяти, если не будет выполнена операция освобождения памяти с помощью оператора delete перед переопределением указателя nums.

```
int *nums = new int[10];
delete[] nums; // Освобождение памяти
nums = new int[11];
```

Работать с этой переменной можно как с обычным массивом, записывать туда значения, изменять их, обращаться по индексам, но самое важное: Пользователь может создать такой массив прямо во время выполнения программы (ввести его размер и работать с массивом такого размера)

```
int size;
cin >> size;
int *nums = new int[size];
```