Два вопроса 1-30 и 31-60
2.
К субботе подготовить план ответов.
5. Лексемы, алфавит — разобрать жёстко.
6. Определение ключевого слова. Примеры и и для чего используется. Коротко. Наспамить много.
7. точки, запятые , точки с запятой, стрелки и тд. Перечислить, кратко за что отвечают. Многие значения одного знака.\
8. Определение, несколько примеров(создать свою переменную). Ниже привести примеры правильного и не правильного названия переменных(вроде 9)
9.Переменные, которые нельзя изменить. Можно назвать лексемой и почему. Может константой быть и функция. Может быть параметр. Почему используется. Хранится в памяти иначе(замена на этапе компиляции). Константный указатель. Документация с++ выжимку.
10. константные строки, как хранятся в памяти. С-строки константные. String константы. Открыть документацию
11. определение. Как то положить на бумагу. Не только в арифметических, но и логических операциях. В теле for тоже есть операнды
12. все типы данных(вообще все). Базовые и ещё short и long как модификаторы. size_t wchar_t — отдельный тип, производный. Int 128(?). Во всех типах данных максимальные значения. IEEE 754
13. Определене переменной. Базовые типы данных в стеке. Что такое стек — область памяти last in first out. Программа работает сверху вниз, при выходе удаляет в обратном порядке. Переменные в куче, объявление, что такое куча. Статическая память и глобальные переменные. Enum, структура. Struct of set. Обращенеие к элементам в стеке, в куче.
14. псевдокод, текстовый, словесный, на яп, блок-схема. Гост, стандарт предприятия, элементы и за что они отвечают. Правильная маленькая блок-схема — пример — без использования элементов языка программирования.
15. выражение — возвращает значение?. Studfiles открыть с впн-ом.
16. Определение. Инверсия, присваивание, умножение, деление, приведение к типу. Взятие указателя, размера, ссылки, взятие-удаление деструктора.
17. Табличка на степике
18. поразрядные — побитовые, отличаются от булевых(и или). Результат выражения — множество нулей и единиц. Не входят логические операции, булевая логика. Арифметические операции. Bool = true когда там любое ненулевое двоичное значение.
19. сдвигает множество нулей и еденицы влево или вправо. Побитовый сдвиг умножает\ делит число на 2. Циклический. Реализация в С++.
20.
21. Приведение типов упомянуть. Перегрузка. Внимание на структуры
22.
23. От меньшего к большему, от целого к вещественному. Почему — написать
24. Выучить наизусть табличку
25. определение потока. Определение потока ввода. Метода которую вадим скидывал? Форматирование — преобработка перед записью в память.
26. понятие потока. Понятие потока ввода-ввывода. Cin pick и т д

27. определение оператора. Простой и составной оператор. Простой - + = += - * . Составной — if for while do while switch case. Примеры и разница.
28. любые ветвления — пропустить кусок кода. Ключевые слова, операции. Перечислить
29. if else
30. goto . Как связан со switch case и почему он не безопасен.
31. Определение цикла. Использование. Какие бывают. Рассказать про for
32, -||- while do while
33. Проблематика бесконечных циклов.
34. break continue. Break на какой уровень выходит. При каких случаях break работает как continue
35. определение массива. Инициализация статического и динамического. Одномерный, двумерный. Побольше про стат массивы и их положение в стеке.
36. определение, зачем и почему. Массив указателей, использование указателей в массивах. Положение указателей в памяти в куче. Скорость обращения к динамич. Массиву меньше чем к статическому.
37. указатели разного порядка. **** на ** чё будет.
38. определение указателя. Инкремент увеличивает не на единицу, а на машинное слово — 4 байта, 8 байт и т д. Арифметика указателей используется в массивах. Разименование адреса. Любой адрес — unsigned long long
39. стат не меняются. Динамич меняются. Способы выделения и освобождения памяти. Где что используется. Табличка отличий со степика написать. Большинство стандартных
40. пример на рекурсии и на цикле. Что быстрее. Проблема — переполнение стека
41. все алгоритмы сортировки. Худший лучший средний случай. Qsort реализация. Запомнить наизусть
42. что такое строка. В языке С — массив символов. String.h. прототип. писать какие типы данных аргументы принимают. В с++ лучше не использовать.
43. - // - . и массив символов и класс стринг. Функции указываем с типами данных. Стринг можно привести к с-строке. Как правильно передать массив символов в стринг?(на 10)ж
44-45. существует struct. Можно через typedef можно нет. Structset не к степени двойки. Вложенные и встроенные струтуры. Область видимости. Можно работать и как с классом(?), но лучше как с .... Инициализация полей, правила. Как можно получить доступ.
45 точно. Какой размер структуры если игнорировать правила инициализирования от меньшего к большему. Что будет если написать pragma pack 1 или pragma pack 59, размер структуры.
46. что такое, чем отличается, область видимости.
В ЧЁМ РАЗНИЦА ПРИ ИНИЦИАЛИЗАЦИИ ЧЕРЕЗ УКАЗАТЕЛЬ И НЕ ЧЕРЕЗ УКАЗАТЕЛЬ, ОБРАЩЕНИЕ ЧЧЕРЕЗ ТОЧКУ И СТРЕЛОЧКУ (44-46)
47. пример объявления структуры через указатель. Структура может хранить указатель на саму себя
48. показать что будет если создать стракт и поле на 1 байт. Имена не конкретному биту, а множеству битов как.

49. локальные внутри области видимости, глобальные вне. Разные типы данных. Статические объекты. Зачем создавать огромный массив в статической памяти.
50 не про auto. Про область видимости. Она в какой-то момент времени её life-cycle закончился при выходе из области видимости. В конце упомянуть, что есть auto.
51. extern , static, глобальные переменные. Примеры использования
52. директивы препроцессора какие кто зачем почему. Подробно про дефайн, какие подводные, взять с документации гугла. Про инклюд подробно. Что можно включать в инклюд.
53. определение. Что это за директивы.
54. учебник по информматике 10 класс. История не интересно. Чем отличается алгоритм от программы и способы описания алгоритмов. Блок-схемы
55. подробно расписывать . Что это зачем это и как работает. О-Нотация по тактам и не по тактам. Мы используем не по тактам. Существуют скрытые константы и примеры. Можно привести в пример умножение матриц и разные его алгоритмы.
56. определение массива, поиска. Поиск переменной, объекта, слова, подстроки и т д. Несколько примеров целочисленного поиска. Бинарный поик написать кодом.
57.
58. операция поиска что такое. А теперь к нечёткому поиску . Ищет вхождение подстроки в строку именно с возможностью дать пользователю ошибится. Рисовать таблицу. Два разных расст. Лев.
59. Алгоритм. Оптимизированное и неоптимизированное. Написать код умножения матриц просто запомнить
60. есть c++ он компилируемый. Что такое коммпиляция. Какими средствами смаке, ручная. Рассмотрим ручную . Флаги чё делают. Подключение нескольких файлов, санитайзеры
61. ручна компиляция библиотек. Как руками скомпилировать и интегрировать
62 тоже.

Вопрос строится фундаментально. Первое что всегда пишем — определение того, на чём строится билет. Если определение неверное — всё из него следующее аннулируется.

писать на компе в LMS
код на листочке 200 задач возможно на компе на solve-е
на листочке — простая
на компе — средне-сложная

19. Операции сдвига (побитовый, циклический)

Все числа в памяти представлены в двоичной форме. Операция побитового сдвига применяется для целочисленных значений и позволяет сдвинуть биты числа влево (<<) или вправо (>>). Синтаксис:

6 >> 1 — сдвиг вправо на один бит. 0110 → 0011. Результат — 3.
6 << 1 — сдвиг влево на один бит. 0110 → 1100. Результат — 12.

При сдвиге в какую-либо сторону, с противоположной стороны на месте сдвинутых появляются нули. Если число имеет в знаковом разряде еденицу (отрицательное число), то при сдвиге влево на месте сдвинутых появляются не нули, а единицы — происходит расширение знака. Для отрицательных чисел выполнение побитового сдвига не рекомендуется — лучше поиграться с преобразованием в беззнаковый.

Сдвиг влево эквивалентен умножению на два. Сдвиг вправо — делению на два.

При сдвиге информация о битах, ушедших за границу разрядности переменной (размер переменной в памяти) теряется полностью. Если это был значимый бит(1) — происходит потеря данных.

Циклический сдвиг — при котором биты, уходящие за границу разрядности не исчезают, а заменяют собой сдвигаемые с противоположной стороны. В C++ можно использовать данную реализацию:

Пусть дано число ***x*** надо совершить циклический сдвиг его битов на величину **d**. желаемый результат можно получить, если объединить числа, полученные при выполнении обычного битового сдвига в желаемую сторону на **d** и в противоположном направлении на разность между разрядностью числа и величиной сдвига. Таким образом, мы сможем поменять местами начальную и конечную части числа.

```
int32 rotateLeft(x, d: int32):
 return (x << d) | (x >>> (32 - d))

int32 rotateRight(x, d: int32):
 return (x >>> d) | (x << (32 — d))
```

20. Операции отношения, логические операции

В C++ существуют операции отношиения и логические операции. В зависимости от операндов, к которым они применяются, операция возвращает значение true или false.

Операции отношения:

- == эквиваленция. Если операнды равны, возвращает true(1), иначе false(0)
- != неравенство. Если операнды не равны, возвращает true(1), иначе false(0)
- < меньше
- > больше
- <= меньше или равно
- >= больше или равно

Логические операции:

- && логическое И. Возвращает true(1), еслы все операнды истинны
- || логическое ИЛИ. Возвращает true(1), если хотя бы один из операндов истиннен
- ! унарное НЕ. Возвращает true(1), если исходная операция возвращала false(0) и наоборот, если исходная false(0), то возвращает true(1).

21. Операция присваивания

Операция присваивания выполняется над двумя операндами и сохраняет значение второго операнда в объект, указанный первым операндом.

Синтаксис: a = 10; b = a.

Если оба объекта имеют арифметические типы, правый операнд преобразуется в тип операнда слева перед выполнением операции.

Существуют составные операторы присваивания, сочитающие операцию присваивания с другими операциями:

- += сложение с присваиванием
- -= вычитание с присваиванием
- *= умножение с присваиванием
- /= деление с присваиванием
- <<= сдвиг влево с присваиванием
- >>= сдвиг вправо с присваиванием
- &= поразрядная конъюнкция с присваиванием
- |= поразрядная дизюнкция с присваиванием
- ^= поразрядное исколючающее ИЛИ с присваиванием

22. Условная трехместная (тернарная) операция

Тернарная операция — операция с тремя операндами. В C++ существует тернарный оператор. Он имеет следующий синтаксис:

выражение ? выражение : выражение

Тернарный оператор работает следующим образом:

- Первый операнд преобразуется в bool
- Если первый операнд является true (1), то выполняется второй операнд
- Если первый операнд является false (0), то выполняется третий операнд

Результатом тернарного оператора является второй или третий операнд.

Если типы второго и третьего операндов не одинаковы, то компилятором вызывается преобразование типов в соответствии со стандартом C++, что может привести к неверной работе программы. Лучше следить за типами вручную.

Следует помнить, что любое значение кроме единицы в переменной типа bool воспринимается как false.

Тернарные операторы можно вкладывать друг в друга, получая многоуровневые условные конструкции.

23. Безопасность преобразования типов

Тип данных — это характеристика значений, которые может принимать переменная или функция.

C++ - язык программирования со строгой типизацией. Это значит, что переменная имеет только один изначально назначенный ей тип данных, и поменять мы его не можем. Но часто возникает необходимость присвоить переменной значение другого типа. Тогда используется преобразование типов. Это перевод значения из одного типа данных в другой.

Если присвоить переменной значение, выходящее за диапозон её типа данных, то компилятор произведёт неявное преобразование значения к соответствующему типу. В арифметических операциях все операнды должны быть одного типа, и компилятор приводит все значения к типу данных с наибольшим диапозоном, участвующим в операции. При этом велика вероятность потери информации, так что этого лучше избегать.

Существует два способа преобразованияя типов: круглые скобки и static_cast. Круглые скобки — унаследованный от C способ, static_cast появился уже в C++ и является более приоритетным, как более безопасный.
Пример:

```
int a = 6;
double b = (double)a;
double c = static_cast<double>(a);
```

Безопасное преобразование типов — при котором не происходит потеря информации. Опасное — при котором происходит потеря информации. Общее правило безопасного преобразования — от меньшего к большему, от целого к вещетсвенному.

Цепочки безопастного преобразования:

bool → char → short → int → double → long double

bool → char → short → int → long → long long
unsigned char → unsigned short → unsigned int → unsigned long
float → double → long double

24. Приоритет операций и порядок вычисления выражений.

В C++ существует приоритет выполнения операций. Если у всех операций в выражении одинаковый приоритет, то почти всегда они выполняются слева направо. Если в выражении операции с разным приоритетом — операции выполняются по убыванию приоритета.

Ссылка на таблицу: https://en.cppreference.com/w/cpp/language/operator_precedence
На русском: https://pvs-studio.ru/ru/blog/terms/0064/
Приоритет операций в C++:

Приоритет	Оператор	Описание
1	`::`	Разрешение области видимости
2	`a++  a--`	Суффиксный/постфиксный инкремент и декремент
	`тип``()  `*`тип`*`{}`	Функциональный оператор приведения типов
	`a()`	Вызов функции
	`a[]`	Индексация
	`.  ->`	Доступ к элементу
3	`++a  --a`	Префиксный инкремент и декремент
	`+a  -a`	Унарные плюс и минус
	`!  ~`	Логическое НЕ и побитовое НЕ
	`(`*`тип`*`)`	Приведение типов в стиле C
	`*a`	Косвенное обращение (разыменование)
	`&a`	Взятие адреса
	`sizeof`	Размер в байтах
	`co_await`	Выражение await (C++20)
	`new  new[]`	Динамическое распределение памяти
	`delete  delete[]`	Динамическое освобождение памяти
4	`.*  ->*`	Указатель на элемент
5	`a*b  a/b  a%b`	Умножение, деление и остаток от деления
6	`a+b  a-b`	Сложение и вычитание
7	`<<  >>`	Побитовый сдвиг влево и сдвиг вправо
8	`<=>`	Оператор трёхстороннего сравнения (C++20)
9	`< <= > >=`	Для операторов отношения < и ≤ и > и ≥ соответственно
10	`== !=`	Операторы равенства = и ≠ соответственно
11	`&`	Побитовое И
12	`^`	Побитовый XOR (исключающее или)
13	`|`	Побитовое ИЛИ (включающее или)
14	`&&`	Логическое И
15	`||`	Логическое ИЛИ
16	`a?b:c`	Тернарный условный оператор
	`throw`	Оператор throw
	`co_yield`	Выражение yield (C++20)
	`=`	Прямое присваивание
	`+=  -=`	Присваивание с сложением и вычитанием
	`*=  /=  %=`	Присваивание с умножением, делением и остатком
	`<<=  >>=`	Присваивание с побитовым сдвигом влево и вправо
	`&=  ^=  |=`	Присваивание с побитовым И, XOR и ИЛИ
17	`,`	Запятая

25. Функции форматированного вывода printf и ввода информации scanf

Поток — непрерывный процесс, который обрпбатывает и выполняет команды. Поток ввода-вывода нужен для получения данных (из файла, от пользователя) и для вывода данных(например, в консоль). При обмене с потоком используется вспомогательный участок памяти — буфер потока. Для работы с потоками ввода и вывода в C необходимо подключить заголовочный файл stdio.h.

Для вывода информации используется функция printf(). Общая форма записи:

printf("СтрокаФорматов", объект1, объект2, ..., объектn);

Строка формата состоит из следующих элементов:

- управляющие символы
- текст, предназначенный для непосредственного вывода
- форматы, предназначенные для вывода значений переменных

Управляющие символы не выводятся на экран, а управляют расположением выводимых символов. Отличительной чертой управляющего символа является наличие обратного слэша '\' перед ним. Основные - /n — перевод строки и /t — горизонтальная табуляция.

Форматы нужны для того, чтобы указывать вид, в котором информация будет выведена на экран. Отличительной чертой формата является наличие символа процент '%' перед ним:

- %d — int
- %u — unsigned int
- %x — int в шестнадцатиричной
- %f — float
- %lf — double
- %c — char
- %s — строка
- %p — указатель

Пример:

```
int x = 5;
printf("У меня %d яблок", x);
```

Для форматированного ввода информации используется функция scanf(). Общая форма записи:

```
scanf ("СтрокаФорматов", адрес1, адрес2,...);
```

В данной функции работаем не переменную, а её адрес!

Строка форматов аналогична функции printf().

Пример:

```
scanf("%f", &y);
```

Функция scanf() является незащищённой, так что для её работы в современных компиляторах необходимо разрешить её использование, добавив в программу строчку
#define _CRT_SECURE_NO_WARNINGS

Существует защищённый вариант — scanf_s().

26. Методы форматированного вывода cout и ввода информации cin

Поток — непрерывный процесс, который обрпбатывает и выполняет команды. Поток ввода-вывода нужен для получения данных (из файла, от пользователя) и для вывода данных(например, в консоль). При обмене с потоком используется вспомогательный участок памяти — буфер потока. Для работы с потоками ввода и вывода в C++ необходимо подключить библиотеку iostream.

Основными функциями для работы с потоками ввода-вывода являются cin и cout. Они находятся в пространстве имён std, поэтому их вызов записывается следующим образом:

std::cin >> a;
std::cout << a << b;

Для операций ввода-вывода переопределены операции поразрядного сдвига:

- >>получить из входного потока
- <<поместить в выходной поток

Для работы с широкими символами (wchar_t) существуют аналоги: wcout и wcin.

Существуют функции-манипуляторы. Функцию — манипулятор потока можно включать в операции помещения в поток и извлечения из потока (<<, >>).
В C++ имеется ряд манипуляторов. Рассмотрим основные:

- endl — очищает буфер потока и переходит на новую строку
- width — устанавливает ширину поля вывода
- precision — устанавливает количество цифр после запятой

У потока ввода также есть функции:
- cin.peek — возвращает следующий символ без извлечения
- cin.ignore(число, символ) — удаляет из потока ввода *число* символов, пока не встретит *символ*.

27. Понятие оператора. Оператор простой и составной, блок оператора

Оператор — это команда, обозначающая определенное математическое или логическое действие, выполняемое с данными (операндами).

В языке C существует следующие группы операторов:

- Условные операторы (if, switch)
- Операторы цикла (while, for, do-while)

- Операторы безусловного перехода (break,continue,goto,return)
- Метки (case,default)
- Операторы-выражения – операторы, состоящие из допустимых выражений.

Блоки – фрагмент текста программы, оформленный фигурными скобками {}. Блок иногда называют составным оператором.

Простым оператором является такой оператор, который не содержит в себе других операторов.

Составной оператор – представляет собой два или более операторов, объединенных с помощью фигурных скобок. Составной оператор иногда называют блоком. Составной оператор рассматривается компилятором как один оператор.

На языке Си любой оператор заканчивается точкой с запятой. Операторы можно условно подразделить на две категории: исполняемые - с их помощью реализуется алгоритм решаемой задачи, и описательные, необходимые для определения типов пользователя и объявления объектов программы, например, переменных.

Исполняемые операторы также можно разбить на две группы: простые и структурированные. В структурированных операторах можно выделить части, которые сами могут выступать в качестве отдельных операторов, а простые операторы на более элементарные разложить не удастся.

К простым операторам относятся: оператор присваивания, оператор-выражение, пустой оператор, операторы перехода (goto, continue, break, return), вызов функции как отдельного оператора.

Структурированные операторы – это операторы ветвления (if), выбора (switch), цикла (for, while, do).

30. Метки и переходы. Оператор выбора (switch-case)

В дополнение к стандартным методам работы с ветвлением в C++ существует оператор **goto**. Он позволяет перейти в конкретное место в коде, заданное **меткой** — идентификатором, состоящим из названия и двоеточия после него. В функции не может быть двух меток с одинаковыми названиями.
Пример:

```
int x = 9;
Loop1:
if(x < 20){
        ++x;
        goto Loop1;
};
```

Switch-case — конструкция для ветвления, позволяющая выбирать между несколькими разделами кода в зависимости от значения **целочисленного** выражения. Она также использует метод переходов по меткам.
Switch-case имеет следующий синтаксис:

```
switch(инициализация, выражение){
        case значение 1:
                //код
                break;
        case значение 2:
                //код
                break;
        default:
                //код
}
```

case и **default** — **метки**. Если выражение в объявлении совпадает со значением **case**, то выполнение кода переходит на эту метку и продолжается в обычном режиме. Для того, чтобы другие случаи case не выполнились, необходимо добавить оператор прерывания **break**. Случай **default** выполняется, если выражение в объявлении не соответствует ни одному case.

Инициализация переменной в объявлении необязательна, при её отсутствии нет необходимости оставлять пустое место и ставить запятую.
Хорошим тоном считается каждый блок case заключать в свою область видимости {}(как в циклах).

В целом использование **goto** и **swicth-case** не рекомендуется. Переход в произвольное место кода нарушает ход следования чтения программы сверху-вниз. С помощью метки возможен переход прямо в центр другого блока кода, цикла, ветвления, что также плохо для понимания. Возникают трудноотслеживаемые ошибки. Компилятору труднее (а иногда — невозможно) оптимизировать такой код.

Однако в некоторых случаях **goto** является хорошим инструментом. Один из примеров — выход из глубоко вложенного цикла.

31. Понятие цикла. Операторы цикла: цикл с заданным числом повторений

Цикл — управляющая конструкция, которая заставляет блок кода повторятся несколько раз. В программировании используются повсеместно для повторения каких-либо инструкций, например — проход по массиву и изменение каждого элемента. В C++ существуют циклы с заданным числом повторений (for), циклы с предусловием (while), и циклы с постусловием (do while).

Цикл с заданным числом повторений в C++ это цикл **for**. Синтаксис цикла **for** при определении имеет следующую форму:

```
for(инициализатор; условие; итерация ){
        //очень важный код;
}
```

Все параметры являются **обязательными** при определении. Поле параметра при надобности можно оставить пустым, но точка с запятой необходимы.
Инициализатор — обычно используется для инициализации итерируемой переменной
Пример:

```
for(int i = 0; ; ){}
```

Условие — условие выхода из цикла. Зачастую в условии используется итерируемая переменная, но это не обязательно. Если возвращает true — цикл продолжается, если false — цикл прерывается.
Пример:

```
for(int i = 0; i < 10; ){}
```

Итерация — действие, которое производится после выполнения одного повторения цикла. Зачастую это изменение итерируемой переменной на некоторый шаг.
Пример:

```
for(int i = 0; i < 10; ++i){
        //очень важный код;
}
```

Последний цикл повторит очень важный код 10 раз. В первой итерации i = 0, в последней итерации i = 9. При проверке на следующее повторение цикла i станет равна 10, и при проверке условия i < 10 будет неверным, что не позволит циклу повториться и программа пойдёт обрабатывать команды дальше.

32. Понятие цикла. Операторы цикла: цикл с предусловием и с постусловием

Цикл — управляющая конструкция, которая заставляет блок кода повторятся несколько раз. В программировании используются повсеместно для повторения каких-либо инструкций, например — проход по массиву и изменение каждого элемента. В C++ существуют циклы с заданным числом повторений (for), циклы с предусловием (while), и циклы с постусловием (do while).

Цикл с предусловием перед выполнением блока кода проверяет на истинность условие. Если оно ложно, то блок не выполняется. Если истинно — то блок кода

выполняется и условие проверяется заново с последующим повторением (или нет) выполнением кода. Таким образом образуется цикл с предусловием. В C++ цикл с предусловием имеет следующий синтаксис:

```
while(условие){
        // код
}
```

Для выхода из такого рода циклов часто кроме условия используется оператор прерывания цикла **break**, который позволяет прямо в цикле прервать его выполнение. Обычно используется в связке с условным оператором **if**.

Цикл с постусловием отличается от цикла с предусловием тем, что блок кода **в любом случае** выполняется 1 раз и после этого при истинности условия выполняется ещё раз.
В C++ имеет следующий синтаксис:

```
do {
        // код
} while (условие)
```

Данные виды циклов повсеместно используются в написании программ. Такие циклы используются, когда неизвестно необходимое количество повторений кода для получения нужного разельтата. Часто цикл с предусловием используется для подсчёта инераций. Для этого заводится переменная-счётчик.
Пример:

```
int number = 100;
int count = 0;
while(number > 10){
        number = number / 2;
        ++count;
}
std::cout << count; // сколько раз нужно число number разделить на 2, чтобы оно стало меньше 10
```

Также стоит упомянуть схему написания всегда истинных циклов с условием.
Пример:

```
while (true){}
```

Для выхода из таких циклов используется оператор прерывания **break**.

33. Понятие цикла. Бесконечные циклы. Проблемы

Цикл — управляющая конструкция, которая заставляет блок кода повторятся несколько раз. В программировании используются повсеместно для повторения каких-либо инструкций, например — проход по массиву и изменение каждого элемента. В C++ существуют циклы с заданным числом повторений (for), циклы с предусловием (while), и циклы с постусловием (do while).

Бесконечный цикл — цикл, условие выхода из которого никогда не выполняется. Если программа захоидт в бесконечный цикл, зачастую это заканчивается ошибкой, чаще всего — переполнением стэка (stack overflow).

При создании программы важно следить, чтобы используемые циклы никогда в условиях программы не могли превратиться в бесконечные.

Примеры бесконечных циклов:

```
int i = 10
while(i>5){
        std::cout << “Этот цикл бесконечен” << std::endl;
}

for(int i = 0; 1; ++i){
        std::cout << i << std::endl;
}
```

34. Понятие цикла. Операторы прерывания и продолжения цикла

Цикл — управляющая конструкция, которая заставляет блок кода повторятся несколько раз. В программировании используются повсеместно для повторения каких-либо инструкций, например — проход по массиву и изменение каждого элемента. В C++ существуют циклы с заданным числом повторений (for), циклы с предусловием (while), и циклы с постусловием (do while).

Операторы перехода:
Оператор **break** завершает выполнение ближайшего внешнего цикла или оператора switch. Оператор **continue** начинает новую итерацию ближайшего внешнего цикла.

Оператор **break** завершает выполнение только одного цикла. Для выхода из нескольких вложенных циклов искользуются другие методы.