1) **Функции ввода-вывода одиночного символа.**  
В C++ для работы с одиночными символами можно использовать методы потоков ввода-вывода:

* Для вывода – метод cout.put(символ).
* Для ввода – метод cin.get(переменная\_для\_символа).  
  Кроме того, в стандартной библиотеке языка (наследуемой от C) есть функции getchar() и putchar(), но в C++ предпочтительнее использовать объекты cin и cout.

2) **Функции ввода-вывода стандартного текстового (символьного) потока.**  
Здесь речь идёт об основных потоках ввода-вывода, таких как:

* std::cin – стандартный поток ввода.
* std::cout – стандартный поток вывода.
* std::cerr – поток для вывода сообщений об ошибках (небуферизуемый).
* std::clog – поток для вывода логов (буферизуемый).  
  Эти потоки обеспечивают базовую работу с текстовыми данными.

3) **Стандартные потоки ввода-вывода символьных данных и строк.**  
Для символьных данных и строк используются те же потоки, что и для ввода-вывода текстовых данных (выше перечисленные cin, cout, cerr, clog). При этом ввод и вывод строк можно осуществлять с помощью оператора >> (форматированный ввод) или методом getline() для чтения целой строки, а вывод – через оператор <<.

4) **Форматированный ввод-вывод символьных данных и строк.**  
Форматирование достигается использованием перегруженных операторов (<< и >>) вместе с манипуляторами из библиотеки <iomanip> (например, setw(), setprecision() и др.). Это позволяет задавать ширину поля, точность, выравнивание и другие параметры вывода. В C-функциях форматирование реализуется через printf() и scanf(), но в C++ предпочтительнее использовать потоки.

5) **Что такое строка? Как задаются строки в программе на C++?**  
Строка – это последовательность символов, которая завершается специальным нулевым символом '\0'. В C++ строки можно задавать двумя основными способами:

* **C-строки:** массивы типа char (например, char str[] = "Hello";).
* **Строки C++:** объекты класса std::string из библиотеки <string>, что упрощает работу со строковыми данными.

6) **Запишите возможные способы начальной инициализации строки.**  
Для C-строк:

* char str[] = "Привет";
* char str[10] = "Привет";
* char \*str = "Привет"; *(но изменять содержимое литерала не рекомендуется)*  
  Для C++-строк:
* std::string str = "Привет";
* std::string str("Привет");

7) **Какой символ стоит в конце строки?**  
В конце строки стоит специальный нулевой символ '\0', который сигнализирует о завершении строки.

8) **Какая нужна вспомогательная библиотека для обработки символов и строк?**  
Для работы с C-строками используются библиотеки:

* <cstring> (или <string.h>) – для работы с функциями копирования, сравнения, вычисления длины и т.д.
* <cctype> – для операций над символами (например, проверка, является ли символ цифрой или буквой).  
  Для C++-строк применяется библиотека <string>.

9) **Какие операции определены для текстовых строк?**  
Для C-строк (массивов символов) функции из <cstring> предоставляют такие операции, как:

* strlen() – вычисление длины,
* strcpy() – копирование,
* strcat() – конкатенация,
* strcmp() – сравнение.  
  Для объектов std::string перегружены операторы: + (конкатенация), ==, <, >, а также определён набор методов: length(), size(), substr(), find(), append() и т.д.

10) **Перечислите операции над символами.**  
Основные операции над отдельными символами включают:

* Присваивание (например, char ch = 'A';)
* Сравнение (операторы ==, <, >, <=, >=)
* Арифметические операции (так как char – это целочисленный тип, можно выполнять операции сложения, вычитания и т.п.)
* Инкремент/декремент (ch++, ch--)
* Функции из <cctype> для проверки типа символа (например, isalpha(), isdigit(), toupper(), tolower()).

11) **Можно ли выполнить присваивание символьной переменной числового значения? Почему?**  
Да, можно. Символьный тип (char) в C++ является целочисленным типом, поэтому при присваивании числового значения оно интерпретируется как код символа согласно таблице ASCII (или другой используемой кодировке).

12) **Допустима ли операция сравнения над символами? Если да, то каким образом определены отношения "больше" и "меньше"?**  
Да, операция сравнения над символами допустима. Сравнение основано на числовых значениях символов (их кодах в таблице ASCII или Unicode). Таким образом, символ с большим числовым значением считается «больше», а с меньшим — «меньше».

13) **Почему символ и строка, состоящая из одного символа, занимают разный объем памяти?**  
Одиночный символ – это просто переменная типа char, которая занимает один байт. А строка – это массив символов, который обязательно содержит завершающий нулевой символ '\0'. Таким образом, строка, содержащая один символ, занимает минимум два байта (один для символа и один для нулевого символа).

14) **Как получить доступ к отдельным символам строки?**  
Если строка задана как массив символов, доступ можно получить через индексирование:

cpp

CopyEdit

char str[] = "Hello";

char ch = str[1]; // получаем второй символ

Для std::string также доступ возможен через перегруженный оператор [] или метод at().

15) **Как можно изменить строку в процессе выполнения программы?**  
Строку можно изменять, если она хранится в изменяемой памяти:

* Для C-строк – если строка объявлена как массив символов, то её можно изменять по индексам (но не изменять строковый литерал, на который указывает указатель).
* Для объектов std::string предусмотрены методы изменения (например, operator[], replace(), append(), insert() и др.).

16) **Почему обращения к строке через ее имя и через указатель эквивалентны?**  
Имя массива (например, строки, объявленной как char str[]) представляет собой адрес первого элемента этого массива. Поэтому обращение через имя строки автоматически подразумевает работу с указателем на её начало.

17) **Почему в качестве параметра функции передается адрес строки, а не сама строка символов?**  
Передача адреса (то есть указателя) позволяет избежать копирования всего массива символов при вызове функции. Это экономит ресурсы и позволяет функции изменять содержимое строки, если это необходимо.

18) **Почему при сравнении строк важен регистр символов?**  
Регистр важен, потому что символы верхнего и нижнего регистра имеют разные числовые коды (например, 'A' и 'a'). Стандартные функции сравнения (например, strcmp()) учитывают эти коды, поэтому даже если символы визуально похожи, они могут считаться разными.

19) **Как сравниваются строки разной длины?**  
При сравнении строк выполняется посимвольное сравнение до первого различия. Если все сравниваемые символы совпадают, то более короткая строка считается «меньшей», поскольку следующему символу в более длинной строке соответствует нулевой символ у короткой.

20) **Какие возможны последствия при обращении к неинициализированной строке?**  
Обращение к неинициализированной строке может привести к неопределённому поведению:

* Использование мусорных данных,
* Нарушению доступа к памяти (segmentation fault),
* Ошибкам логики программы.

21) **Почему при формировании строки без использования стандартных функций необходимо дописывать символ конца строки? Почему этого не требуется при считывании строк с клавиатуры?**  
При ручном формировании строки программист должен явно указать конец строки, добавив символ '\0', чтобы функции могли корректно определить границу строки. При считывании с клавиатуры стандартные функции ввода (например, cin.getline()) автоматически добавляют завершающий нулевой символ.

22) **Для чего предназначена функция strcpy() и в какой библиотеке она определена?**  
Функция strcpy() предназначена для копирования одной C-строки в другую, включая завершающий нулевой символ. Она определена в библиотеке <cstring> (или <string.h> в стандарте C).

23) **Что выполняет функция strcmp()?**  
Функция strcmp() сравнивает две C-строки посимвольно. Если строки идентичны, возвращается 0; если строки различаются, возвращается отрицательное или положительное значение в зависимости от лексикографического порядка.

24) **Что возвращает функция strlen()?**  
Функция strlen() возвращает количество символов в строке до встреченного нулевого символа '\0' (то есть длину строки без учёта завершающего символа).

25) **Приведите простейший алгоритм копирования одной строки в другую.**  
Простейший алгоритм копирования C-строки выглядит так:

cpp

CopyEdit

while (\*src != '\0') {

\*dest = \*src;

dest++;

src++;

}

\*dest = '\0';

Здесь src – указатель на исходную строку, а dest – указатель на буфер назначения.

26) **Почему в С++ не выполняется операция прямого присваивания значения строке?**  
В случае C-строк (массивов символов) операция присваивания не определена, так как массивы не являются полноценными объектами, и их адреса фиксированы в памяти. Присваивание массивов подразумевает копирование всех элементов, что не поддерживается напрямую. Вместо этого используют функции вроде strcpy() или применяют класс std::string, который перегружает оператор присваивания.