21. Строки.

План ответа:
1 понятия «строка» и «строковый литерал»;
2 определение переменной-строки, инициализация строк;
3 ввод/вывод строк (scanf, gets, fgets, printf, puts);
4 функции стандартной библиотеки для работы со строками (strcpy, strlen, strcmp и др.);
5 идиомы обработки строк.

1 Строка – это последовательность символов, заканчивающаяся и включающая первый нулевой символ
  Строковый литерал – последовательность символов, заключенных в двойные кавычки.
	Строковый литерал рассматривается компилятором как массив элементов типа char.
	Когда компилятор встречает строковый литерал из n символов, он выделяет n+1 байт памяти,
	которые заполняет символами строкового литерала и завершает нулевым символом.
	char str[] = “String for test”;

	Строка – массив символов, для доступа к элементу строки может использоваться операция индексации
	Массив, который содержит строковый литерал, существует в течение всего времени выполнения программы.
	В стандарте сказано, что поведение программы не определено при попытке изменить строковый литерал.
	Обычно строковые литералы хранятся в read only секции.

2 Определение переменной-строки, которая может содержать до 80 символов обычно выглядит следующим образом:
	#define STR_LEN 80
	...
	char str[STR_LEN+1];

Инициализация строковых переменных
	char str_1[] = {‘J’,‘u’,‘n’, ‘e’, ‘\0’};
	char str_2[] = “June”;
	char str_3[5] = “June”;

	char str_4[3] = “June”;
	// error: initializer-string for array of chars is too long

	char str_5[4] = “June”;
	// str_5 не строка!

3 Стандартные функции ввода-вывода строк:
	Ввод
		scanf(“%s”,str), fscanf(f,”%s”,str);
		// символы-разделители строк – в том числе пробелы и табуляция.
		Считывание идёт в заранее выделенный буфер.
		Программист сам отслеживает ограничения на длину.

		gets(buf), fgets(buf, n_max, f);
		// считывает строку вплоть до символа перевода строки или конца файла (записывая и его, а после него – терминирующий нуль).
	Вывод
		printf, fprintf // вывод строки без её перевода
		puts, fputs // вывод с переводом строки

4 функции стандартной библиотеке
	strcpy
		char* strcpy(char *s1, const char *s2);
		Вместо функции strcpy безопаснее использовать функцию strncpy
		char* strncpy(char *s1, const char *s2, size_t count);
		Пример
			strncpy(dst, src, sizeof(dst) - 1);
			dst[sizeof(dst) - 1] = '\0';

strlen

```
size_t strlen(const char *s);
```

strcmp

```
int strcmp(const char *s1, const char *s2);
        значение < 0, если s1 меньше s2
        0, если s1 равна s2
        значение > 0, если s1 больше s2
```

Строки сравниваются в лексикографическом порядке.
Функция strcmp сравнивает символы, сравнивая значения кодов, которые представляют эти символы.

```
int strncmp(const char *s1, const char *s2, size_t count);
```

другие

```
char* strdup(const char *s); // НЕ c99
char* strndup(const char *s, size_t count); // НЕ c99
int sprintf(char *s, const char *format, ...);
int snprintf(char *s, size_t num, const char *format, ...); // c99
char* strtok(char *string, const char *delim);  //выполняет поиск лексем в строк
char *strchr (const char *str, int ch); //выполняет поиск первого вхождения символа symbol в строку string.
```

Перевод числа в строку

```
#include <stdlib.h>
// Семейство функций (atoi, atof, atoll)
long int atol(const char* str);
// Семейство функций (strtoul, strtoll, ...)
long int strtol(const char* string, char** endptr, int basis);
```

5 Особенности обработки строк:

Выход за пределы выделенного буфера строки — крайне нежелательный эффект, не отслеживается компилятором и ложится на плечи программиста.

Терминирующий нуль при преобразованиях строк необходимо сдвигать, а также обеспечивать его нахождение в буфере

Пробег по строке с помощью указателей также должен быть аккуратным, использование вне пределов буфера чревато последствиями.