Лекция 10. Символьные строки и строковые функции. Часть 1.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Содержание

- Создание и использование строк
- Символьные массивы и символьные указатели
- Применение строковых и символьных функций из библиотеки Си и создание собственных строковых функций
- Разбор функций ввода строк, анализ.

Ввод-вывод (I/O)

Операции **ввода-вывода** являются базовым процессом внутри которого рассматривается **корректная** передача информации от внешнего мира к обработчику информации и наоборот.
Ввод – данные/сигнал **получаемые (input)** обработчиком информации.
Вывод – данные/сигнал **посланные (output)** обработчиком информации.

- stdio.h (от англ. standard input/output header — стандартный заголовочный файл ввода-вывода)

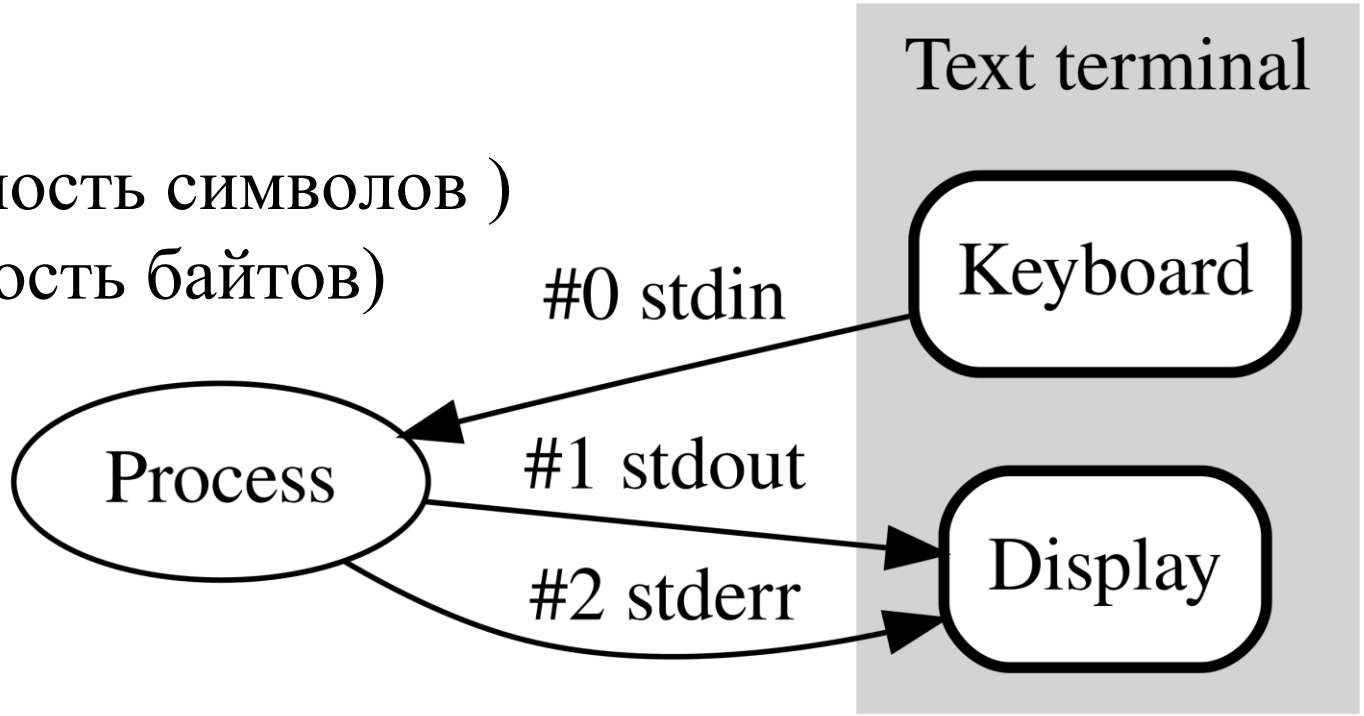
Поток данных

Под **потоком данных** понимается набор информации, выходной или входной от:

- Устройства (Терминал)
- Файла\Сокета

Потоки так же могут быть:

- Текстовым (Последовательность символов)
- Бинарным (Последовательность байтов)



Поток-файлы и перенаправления в Си

Файл – именованная область долговременной памяти, где хранится информация.

Перенаправление позволяет использовать программы, предназначенные для обработки ввода с клавиатуры, с файлами. Для этого в программе должна предприниматься проверка на предмет достижения конца файла.

```
./program < filein
./program > fileout
./program < filein > fileout
```

Символьная строка (Си-style)

Символьная строка в языке Си – это последовательность из одного или большего количества символов, которая заканчивается управляющим символом '\0';
'\0' – называют нулевым символом строки

Символьные строковые литералы

Стандарт ANSI Си, выполняется конкатенация строковых литералов, если они отделены друг от друга ничем, кроме пробельных символов.

```
char stringf[50]="Это""длинная""строка""символов.";
char string[50]="Это длинная строка символов.";
```

Завершающий символ '\0' автоматически добавляется компилятором.

Хранение строковых литералов

```
#include <stdio.h>
int main(){
    printf("%s. %p, %c\n", "Мы", " - " ,
*"космические бродяги");
}
```

```
 Мы. 00007ff60b1f9000, к
```

Массивы символьных строк (1)

Инициализация

```
const char ml[40] = "Постарайтесь уложиться в одну
строку.";
const char ml[40] = {'П','о','с','т','а','р','а',
'й','т','е','с','ь',' ','у','л','о','ж','и’,'т’,
'ь','с','я',' ','в',’ ','о','д','н','у',' ',
'с','т’, 'р','о','к','у','.','\0'};
```

Обратите внимание на завершающий нулевой символ. Без него вы получите символьный массив, а не *строку Cu-style*.

При указании размера массива убедитесь, что количество элементов, по меньшей мере, на единицу больше длины строки (не забывайте о нулевом символе).

Массивы символьных строк (2)

Инициализация

```
const char m2[] = "Если вам не о чем думать,
вообразите что-нибудь.";
```

Почему в это случае мы может позволить компилятору определять размер массива?

```
char car[10] = "Луна";
(car == &car[0]) = ? (*car == 'Л') = ?
(*(car+1)== 'у') = ? (car[1] == 'у') = ?
```

Массивы символьных строк (3)

Инициализация

```
const char m2[] = "Если вам не о чем думать,
вообразите что-нибудь.";
```

Почему в это случае мы может позволить компилятору определять размер массива?

```
char car[10] = "Луна";
(car == &car[0]) = 1 (*car == 'Л') = 1
(*(car+1)== 'у') = 1 (car[1] == 'у') = 1

const char ar1[] = "Что-то указывает на меня.";
const char *pt1 = "Что-то указывает на меня.";
```

Тем не менее эти формы не идентичны. Почему?

Символьные массивы и указатели (1)

```
const char ar1[] = "Что-то указывает на меня.";
```

Запись в форме массива (ar1[]) приводит к размещению в памяти компьютера массива из 26+1 элементов. Каждый элемент инициализируется соответствующим символом строкового литерала.

```
gcc -E main.c -o main.ii && gcc -S main.ii -o main.s
```

Переменные в статической часть исполняемого файла.

```
        .file   "main.c"
        .text
        .def    __main; .scl    2;      .type   32;     .endef
        .globl  main
        .def    main;   .scl    2;      .type   32;     .endef
        .seh_proc       main
```

Символьные массивы и указатели (2)

Из предыдущего слайда следует, что память под значение заключенное в двойные кавычки, выделяется только после того, как программа начнет выполнение.

Имя	const char[]	-	-	-	-	-	-	-	-	-	-	-	-	-	-		
Адрес	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0F	0x0F	0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17	0x18
Значение	Ч	т	о	-	т	о		у	к	а	з	ы	в	а	е	т	
Индекс	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Имя	-	-	-	-	-	-	-	-	-	const char ar1[]	-	-	-	-	-	-	-
Адрес	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F	0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27	0x28	0x29
Значение	н	а		м	е	н	я	.	/0	Ч	т	о	-	т	о		У
Индекс	17	18	19	20	21	22	23	24	25	0	1	2	3	4	5	6	7
Имя	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Адрес	0x2A	0x2B	0x2C	0x2D	0x2E	0x2F	0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x3A
Значение	к	а	з	ы	в	а	е	т		н	а		м	е	н	я	
Индекс	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Символьные массивы и указатели (3)

```
const char ar1[] = "Что-то указывает на
меня.";
printf("%c %p\n",*(ar1), ar1+1);
*(ar1) = 'К';
ar1++;
printf("%c %p",*(ar1), ar1);
```

результат?

Символьные массивы и указатели (4)

```
const char ar1[] = "Что-то указывает на
меня.";
printf("%c %p\n",*(ar1), ar1+1);
*(ar1) = 'к';
ar1++;
printf("%c %p",*(ar1), ar1);
```

```
Ч 000000C13FFFF821
```

Символьные массивы и указатели (5)

```
const char *pt1 = "Что-то указывает на меня.";
```

Запись в форме указателя (*ar1[]) также приводит к тому, что в статической памяти под строку резервируются 26 элементов.

```
gcc -E main.c -o main.ii && gcc -S main.ii -o main.s
```

Переменные в статической часть исполняемого файла.

```
    .file   "main.c"
…
.LC0:
    .ascii "\227\342\256-\342\256
\343\252\240\247\353\242\240\245\342 \255\240
\254\245\255\357.\0"
…
```

Символьные массивы и указатели (6)

Из предыдущего слайда следует, после того как программа начнет выполнятся она выделяет в памяти место под переменную типа указателя pt1 и сохраняет в ней адрес строки.

Имя	`const char *pt1`	const char[]	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Адрес	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0F	0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17
Значение	0x08	Ч	т	о	-	т	о		у	к	а	з	ы	в	а	е
Индекс	-	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Имя																
Адрес	0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F	0x20	0x21	0x22					
Значение	т		н	а		м	е	н	я	.	/0					
Индекс	15	16	17	18	19	20	21	22	23	24	25					

Символьные массивы и указатели (7)

```
const char * pt1 = "Что-то указывает на меня.";
printf("%p %c\n",pt1,*(pt1));
pt1++;
printf("%p %c\n",pt1,*(pt1));
*(--pt1) = 'K';
```

результат?

Символьные массивы и указатели (8)

```
const char * pt1 = "Что-то указывает на меня.";
printf("%p %c\n",pt1,*(pt1));
pt1++;
printf("%p %c\n",pt1,*(pt1));
*(--pt1) = 'К';
```

```
00007FF761164000 Ч
00007FF761164001 т
```

Различия массивов символов и указателей(1)

```
int i=7, j=7;
char ci[] = "Я люблю язык Си!";
const char *python = "Я люблю язык Python!";
while (ci[i]!='\0') putchar(ci[i++]);
putchar('\n');
while (python[j]!='\0') putchar(python[j++])
putchar('\n');
```

```
язык Си!
язык Python!
```

Различия массивов символов и указателей(2)

```
char ci[] = "Я люблю язык Си!";
const char *python = "Я люблю язык Python!";
int i=7, j=7;
while (*(ci+i)!='\0')putchar(*(ci+i++));
putchar('\n');
while (*(python+j)!='\0’)
    putchar(*(python+j++));
putchar('\n');
```

```
язык Си!
язык Python!
```

Различия массивов символов и указателей(3)

```
char ci[] = "Я люблю язык Си!";
const char *python = "Я люблю язык Python!";
int i=7, j=7;
while (*(i+ci)!='\0') putchar(*(i+ci++));
putchar('\n');
while (*(j+python)!='\0')
    putchar(*(j+python++));
putchar('\n');
```

Результат?

Различия массивов символов и указателей(4)

```
char ci[] = "Я люблю язык Си!";
const char *python = "Я люблю язык Python!";
int i=7, j=7;
while (*(i+ci)!='\0') putchar(*(i+ci++));
putchar('\n');
while (*(j+python)!='\0')
    putchar(*(j+python++));
putchar('\n');
```

```
язык Python!
```

Различия массивов символов и указателей(4)

```
char ci[] = "Я люблю язык Си!";
const char *python = "Я люблю язык Python!";
ci = python;
or
python = ci;
Что станет с ci? Или что станет с python?
```

Различия массивов символов и указателей(5)

```
int i=0;
char ci[] = "Я люблю язык Си!";
const char *python = "Я люблю язык Python!";
python = ci; ci = python;
while (ci[i]!='\0') putchar(ci[i++]);
putchar('\n');
while(*(python)!='\0') putchar(*(python++));
```

```
Я люблю язык Си!
Я люблю язык Си!
```

Различия массивов символов и указателей(6)

Элементы массива являются *переменным* и (если только массив не объявлен как ***const***), но *имя* массива - это не *переменная*.

```
char * word = "дело";
word[2] = 'п';
```

Согласно текущему стандарту Си, поведение в этом случае не определено. Такой оператор может, например, привести к ошибке доступа в память. Причина связана с тем, что, как упоминалось ранее, компилятор может выбрать вариант представления всех идентичных строковых литералов в виде единственной копии в памяти.

Различия массивов символов и указателей(7)

```
#define SLEN 40 #define LIM 5
const char *mytalents[LIM]= {
"Мгновенное складывание чисел",
"Точное умножение", "Накапливание данных",
"Исполнение инструкций с точностью до буквы",
"Знание языка программирования С"};
char yourtalents[LIM][SLEN] = {
"Хождение по прямой",
"Здоровый сон", "Просмотр телепередач",
"Рассылка писем", "Чтение электронной почты"
};
```

Различия массивов символов и указателей(8)

```
int i ; puts("Сравним наши таланты.");
printf ("%-52s %-25s\n", "Мои таланты", "Ваши
таланты");
for (i = 0; i < LIM; i++)
printf("%-52s %-25s\n", mytalents[i],
yourtalents[i]);
printf("\nразмер mytalents: %zd, размер
yourtalents: %zd\n", sizeof(mytalents),
sizeof(yourtalents));
```

Различия массивов символов и указателей(9)

```
Сравним наши таланты.
Мои таланты                                    Ваши таланты
Мгновенное складывание чисел                   Хождение по прямой
Точное умножение                               Здоровый сон
Накапливание данных                            Просмотр телепередач
Исполнение инструкций с точностью до буквы     Рассылка писем
Знание языка программирования С                Чтение электронной почты

размер mytalents: 40, размер yourtalents: 200
```

Имя	mytalents [0]	mytalents [1]	mytalents [2]	mytalents [3]	mytalents [4]
Адрес	0x08	0x10	0x18	0x20	0x28

Имя	yourtalents [0][0]	yourtalents [1][0]	yourtalents [2][0]	yourtalents [3][0]	yourtalents [4][0]
Адрес	0x30	0x58	0x80	0xA8	0xD0

Указатели и строки

```
const char * mesg = "Не позволяйте себя запутать!";
const char * copy;
copy = mesg;
printf("%s\n", copy);
printf("mesg = %s; &mesg = %p; value = %p\n", mesg,
&mesg, mesg);
printf("copy = %s; &copy = %p; value = %p\n",copy,
&copy, copy);
```

```
Не позволяйте себя запутать!
mesg = Не позволяйте себя запутать!; &mesg = 000000EE1B9FFE18;
 value = 00007FF70C0A4000
copy = Не позволяйте себя запутать!; &copy = 000000EE1B9FFE10;
 value = 00007FF70C0A4000
```

Ввод строк (1)

```
char *name;
scanf("%s", name);
```

Как поведёт себя компилятор? Программа?

Ввод строк (2)

Не следует ожидать, что компьютер подсчитает длину строки во время ее чтения и выделит необходимое пространство. Компьютер не будет делать это.

```
char name[81];
scanf("%s", name);
```

Теперь name представляет собой адрес зарезервированного блока памяти размером 81 байт.

Ввод строк: функция gets() (1)

Функция `scanf()` и спецификатор `%s` обеспечивают считывание только одного слова.

Однако, возникают задачи в которых нужно считывать **всю строку**, а не одно слово.

Долгие годы для такой цели служила функция `gets()`.

Функция читает всю строку вплоть до символа новой строки, отбрасывает этот символ и сохраняет остальные символы, добавляя в конец строки '\0'.

Для вывода строки лучше использовать функцию `puts()`, которая отображает строку с добавлением в конце символа '\0'.

Ввод строк: функция gets() (2)

```
char * gets ( char * str );
```

Ввод строк: функция gets() (3)

```
#define STLEN 81
char words[STLEN];
puts("Введите строку.");gets(words);
printf ("Ваша строка, выведенная дважды:\n");
printf("%s\n", words);puts(words);puts("Готово.");
warning: call to 'gets' declared with attribute
warning: Using gets() is always unsafe - use
```

```
Введите строку.
Привет
Ваша строка, выведенная дважды:
Привет
Привет
Готово.
```

```
instead [-Wattribute-
warning] << Компилятор
```

Ввод строк: функция gets() (4)

Проблема функции в том, что она не осуществляет проверку, умещается ли вводимая строка в массив или нет. Функция имеет лишь один аргумент, *указатель* позволяет определить начало строки, но не ее размер.

Если строка ввода окажется слишком длинной, возникнет переполнение буфера — другими словами, символы переполнят предназначенное для них целевое пространство. Лишние символы могут просто попасть в неиспользуемую память и привести к проблемам, которые проявятся не сразу, или же они могут перезаписать другие данные в программе. И это — отнюдь не все возможные варианты.

Ошибка: segmentation fault

```
Segmentation fault: 11
Ошибка сегментации: 11
```

В системе Unix такое сообщение указывает на то, что программа попыталась получить доступ в намять, которая не была для нее выделена.

В системах MS-DOS такое сообщение вовсе может не отображаться или ее отображение носит плавающий характер (***Гейзенбаг***).

Отказ от функции gets()

Комитет, создавший стандарт C99, также опубликовал обоснование стандарта. В этом обосновании были признаны проблемы, связанные с использованием функции `gets()`, и применять ее не рекомендовалось. Тем не менее, комитет по созданию стандарта C11 придерживался более строгих взглядов и исключил функцию `gets()` из стандарта.

Ввод строк: функция fgets() (1)

Альтернативой функции `gets()` это функция `fgets()`, которая обладает несколько более сложным интерфейсом и по-другому обрабатывает вводимые данные.

- Она принимает второй аргумент, задающий n количество символов для чтения. Тогда функция `fgets()` прочитает n-1 символов или будет читать до появления символа '\n' в зависимости от того, что произойдёт раньше.

Ввод строк: функция fgets() (2)

- Если функция `fgets()` сталкивается с символом новой строки, она сохраняет его в строке, в отличие от функции `gets()`, которая отбрасывает его.
- Функция `fgets()` принимает третий аргумент, указывающий файл, из которого должно производиться чтение. Для чтения с клавиатуры в качестве этого аргумента используется **stdin** (от standard input - стандартный ввод); этот идентификатор определен в stdio.h.

Ввод строк: функция fgets() (3)

```
char * fgets ( char * str,
int num,
FILE * stream );
```

Вывод строк: функция fputs() (1)

Функция `fputs()` принимает второй аргумент, указывающий на файл, в который должна производится запись. Функция не добавляет автоматически символ '\n'. Для вывода на дисплей можно использовать аргумент **stdout** (от standard output - стандартный вывод).

```
int fputs ( const char * str,
            FILE * stream );
```

Пример: fgets() и fputs() (1)

```
char words[STLEN];
char * mes = "Ваша строка, выведенная дважды (с
помощью puts (), а затем fputs ()) :\n";
puts("Введите строку."); fgets(words, STLEN,
stdin);
printf(mes); puts(words); fputs(words, stdout);
puts("Введите еще одну строку.");
fgets(words, STLEN, stdin); printf(mes);
puts(words);
fputs(words, stdout); puts("Готово."); return 0;
```

Пример: fgets() и fputs() (2)

```
Введите строку.
Вездесущий
Ваша строка, выведенная дважды (с помощью puts (), а затем fputs ()) :
Везде
ВездеВведите еще одну строку.
Ваша строка, выведенная дважды (с помощью puts (), а затем fputs ()) :
сущий
сущийГотово.
```

Пример: fgets() и fputs() (3)

```
#define STLEN 10
char words[STLEN];
puts("Введите строки (или пустую строку
для выхода из программы):");
while (fgets(words, STLEN, stdin) != NULL
&& words[0] != '\n')
    fputs(words, stdout);
puts("Готово.");
```

Пример: fgets() и fputs() (4)

```
Введите строки (или пустую строку для выхода из программы):
Функция fgets () возвращает указатель на char. Если все
проходит нормально, она просто возвращает тот ж е адрес,
который был ей передан в первом аргументе. Однако если
функция встречает конец файла, она возвращает специальный
указатель, называемый нулевым.
Функция fgets () возвращает указатель на char. Если все
проходит нормально, она просто возвращает тот ж е адрес,
который был ей передан в первом аргументе. Однако если
функция встречает конец файла, она возвращает специальный
указатель, называемый нулевым.
^Z
Готово.
```

Плюсы и минусы функции fgets() (1)

Функция `fgets()` сохраняет символ новой строки, порождает проблему, но и предоставляет дополнительную возможность.

\+ Получая символ '\n' можно судить о том, прочитана вся строка или нет.

\- Сохранение символа '\n' в виде части строки может быть нежелательным.

Плюсы и минусы функции fgets() (2)

Если нужно избавится от символа '\n':

```
while (str[i] != '\n') i++;
str[i] = '\0';
```

Если в строке по прежнему остались символы:

```
while (getchar() != '\n') continue;
```

Ввод строк: функция gets_s() (1)

```
char* gets_s( char* str, rsize_t n );
```

Функция представляет собой не обязательное расширение библиотеки Си.

- Функция просто выполняет чтение из стандартного ввода, поэтому она не нуждается в третьем аргументе.
- Если функция считывает символ новой строки, то отбрасывает его, а не сохраняет.

Ввод строк: функция gets_s() (2)

```
char* gets_s( char* str, rsize_t n );
```

- Если прочитает максимальное количество символов, и символ `'\n'` отсутствует, она предпринимает несколько действий. Функция устанавливает первый символ строки в нулевой символ. Затем она читает и отбрасывает последующий ввод, пока не встретится `'\n'` или EOF. Наконец, функция возвращает NULL. Она вызывает зависящую от реализации функцию “обработчика” (или же выбранную вами функцию), которая может привести к выходу из программы или прекращению ее работы.

Сравнение трёх функций

```
char words[10];
gets(), fgets() и gets_s() – сравним;
< Экзамен
    ?       ?       ?
< Кафедра вычислительных систем
    ?       ?       ?
< Очень важная информация которую мы вводим.
    ?       ?       ?
```

Сравнение трёх функций (Ответы)

```
char words[10];
gets(), fgets() и gets_s() – сравним;
< Экзамен
    √       √       √
< Кафедра вычислительных систем
    x       √       √(end)
< Очень важная информация которую мы вводим.
    x       √    x
```

Ввод строк: функция scanf() (1)

```
int scanf ( const char * format, ... );
```

Функция больше ориентирована на “получение слова”, а не на “получение строки”.

- Если задан формат %s, строка продолжается до следующего (не включая его) ' ', (' ', '\n', '\t’).
- Если указана ширина поля, как в %10s, функция читает до получения 10 символов или до появления первого ' ', в зависимости от того, что произойдет раньше.

Ввод строк: функция scanf()

```
#define STLEN 11
char name1[STLEN], name2[STLEN];
int count;
printf("Введите два имени.\n");
count = scanf("%5s %10s", name1, name2);
printf ("Прочитано %d имени: %s и %s.\n",
count, name1, name2);
```

Ввод строк: функция scanf()

```
>Введите два имени.
<Джейн Покер
>Прочитано 2 имени: Джейн и Покер.
>Введите два имени.
<Иван Мительшпиль
>Прочитано 2 имени: Иван и Мительшпил.
>Введите два имени.
<Вениамин Каверин
>Прочитано 2 имени: Вениа и мин.
```

Задачи на практику

Читать: Глава 11 *Символьные строки и строковые функции* (стр. 419-440)

Практических заданий на этой неделе нет.

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. Кафедры вычислительных систем

Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)