3. Обработка строк

3.1. Стандартная библиотека

```
char *strcat(char *s1, const char *s2)
char *strncat(char *s1, const char *s2, size_t n)
const char *strchr(const char *s, int c)
const char *strrchr(const char *s, int c)
int strcmp(const char *s1, const char *s2)
int strncmp(const char *s1, const char *s2, size_t n)
char *strcpy(char *s1, const char *s2)
char *strncpy(char *s1, const char *s2, size_t n)
```

3.1. Стандартная библиотека

size_t strcspn(const char *s1, const char *s2)

size_t strspn(const char *s1, const char *s2)

size_t strlen(const char *s)

const char *strpbrk(const char *s1, const char *s2)

const char *strstr(const char *s1, const char *s2)

char *strtok(char *s1, const char *s2)

3.2. Ввод строки

gets(char *) – ввод строки вместе с кодом ‘\n’;

не контролирует размер памяти

scanf(“%Ls”, buf) – ввод стоки длиной не более L символов (размер памяти – L + 1);

не вводит пробелы и символ ‘\n’

scanf(“%L[^\n]”, buf) – ввод строки длиной не более L символов

вводит любые символы

во входном потоке остается ‘\n’

3.2. Ввод строки

```
char buf[L + 1], c;
int  n;
n = scanf(“%L[^\n]%c”, buf, &c);
```

позволяет удалить из входного потока символ ‘\n’

3.3. Варианты ввода данных

n = scanf(“%L[^\n]%c”, buf, &c);

1. Вводится пустая строка

входной поток: 

Результат:

n = 0

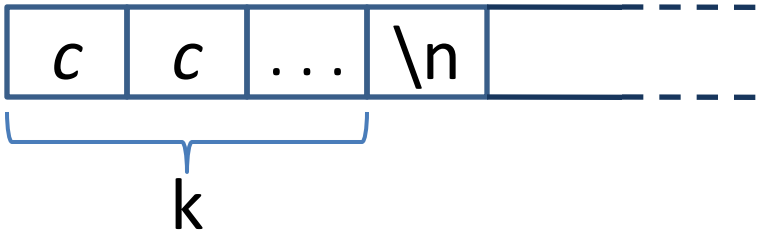
buf и c не меняют своего содержимого

входной поток: 

3.3. Варианты ввода данных

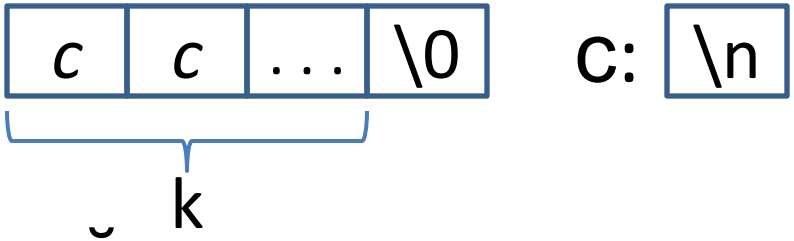
n = scanf(“%L[^\n]%c”, buf, &c);

2. Вводится срока длиной k <= L

входной поток: 

Результат:

n = 2

buf: 

c:

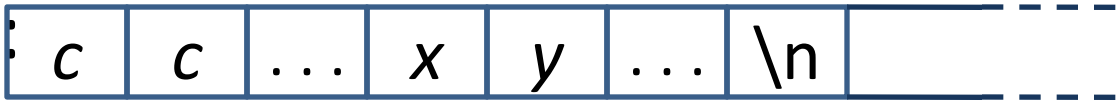
входной поток: 

3.3. Варианты ввода данных

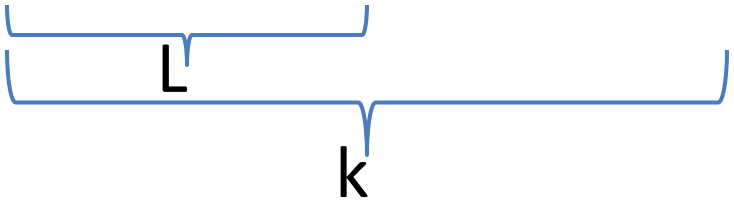
n = scanf(“%L[^\n]%c”, buf, &c);

3. Вводится строка длиной k > L

входной поток:

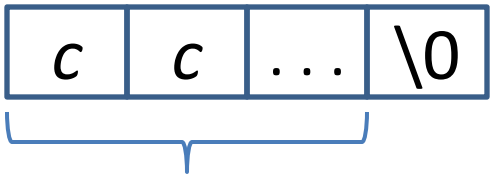


Результат:



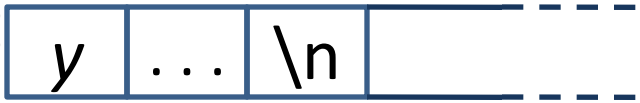
n = 2

buf:



c:

входной поток:



3.4. Алгоритм ввода

цикл {

ввести строку: n = scanf(. . .);

анализ n:

n == -1:

освободить память, результат = NULL

n == 0:

удалить из входного потока ‘\n’

3.4. Алгоритм ввода

n == 2:

проверить c == ‘\n’

да – присвоить n = 0;

нет – вернуть c в поток

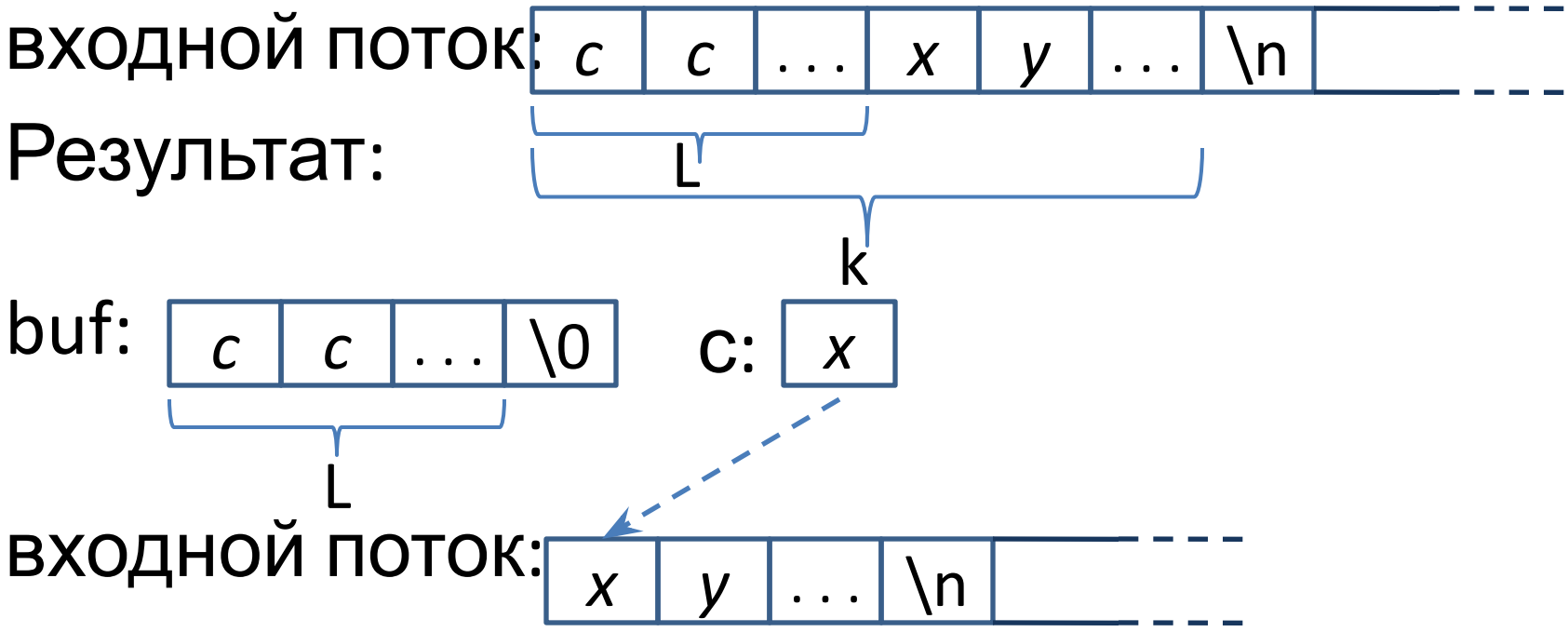
сформировать результирующую строку

} пока n > 0

3.5. Коррекция входного потока

n = scanf(“%L[^\n]%c”, buf, &c);

ungetc(c, stdin);



3.6. Формирование строки

```
char *ptr = (char *)malloc(1);
int len = 0;
*ptr = ‘\0’;
n = scanf(“%L[^\n]%c”, buf, &c);
len += strlen(buf);
ptr = (char *)realloc(ptr, len + 1);
strcat(ptr, buf);
```

3.7. Реализация алгоритма

```
char *getstr()
{
	char *ptr = (char *)malloc(1);
	char buf[81];
	char c;
	int n, l = 0;
	*ptr = '\0';
	do{
		n = scanf("%10[^\n]%c", buf, &c);
		if(n < 0){
			free(ptr);
			ptr = NULL;
		}
		else
```

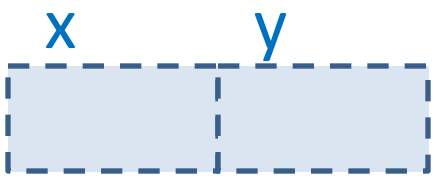
3.7. Реализация алгоритма

```
            if(n == 0)
                scanf("%c", &c);
            else {
                if(c == '\n')
                    n = 0;
                else
                    ungetc(c, stdin);
                l += strlen(buf);
                ptr = (char *) realloc(ptr, l + 1);
                strcat(ptr, buf);
            }
    } while(n > 0);
    return ptr;
}
```

4. Структуры

4.1. Определение структуры

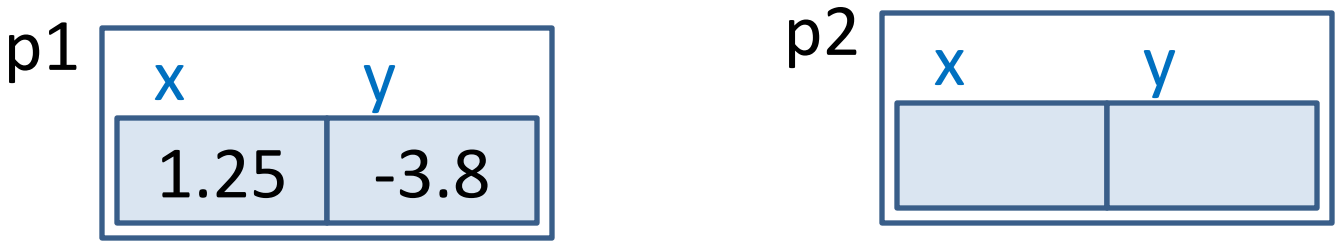
```
struct имя_структуры {
    тип  имя, … ;
    тип  имя, … ;
   . . .
};
struct  Point {
     double  x,  y;
};
```



4.2. Определение переменных

struct *имя_структуры имя_переменной*
= { *значение_1*, *значение_2*, . . . } ;

struct Point p1 = {1.25, -3.8}, p2;



4.3. Определение массива

struct *имя_структуры имя_массива*
[*количество*]
= {
{*значение_01*, *значение_02*, . . . },
{*значение_11*, *значение_12*, . . . },
. . .
};

struct Point pp[3] =
{{1, 1}, {2,2}, {1,2}};

pp

x	y
1.0	1.0
2.0	2.0
1.0	2.0

4.4. Операции со структурами

```
struct Point p1 = {1.5, 2.8};
```

Копирование структуры

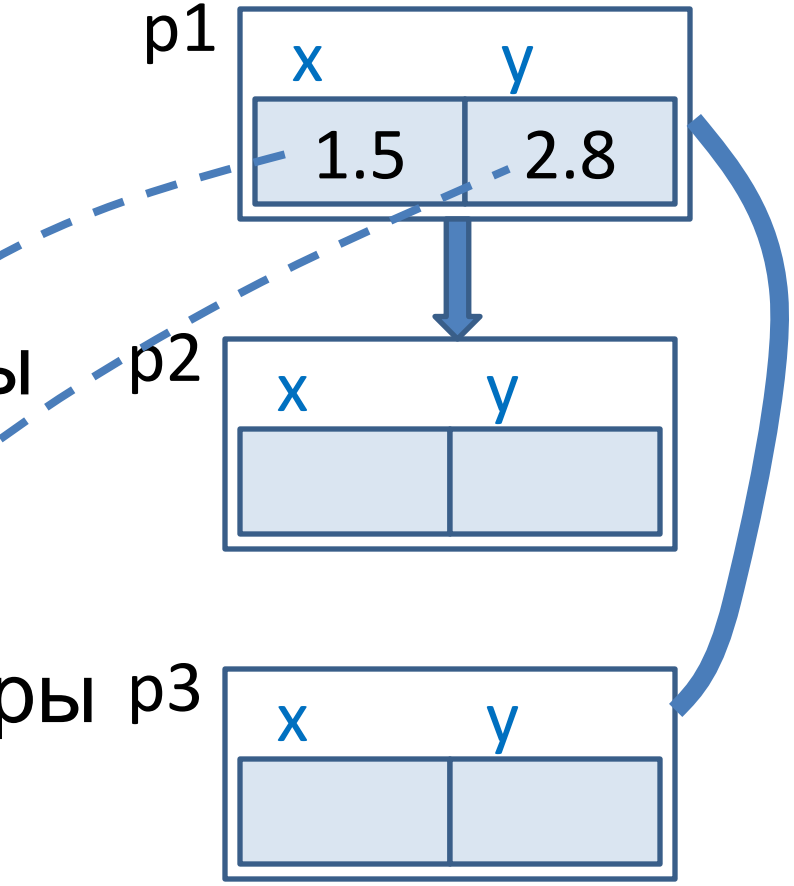
```
struct Point p2 = p1;
```

Присваивание структуры

```
struct Point p3;
p3 = p1;
```

Разыменование структуры

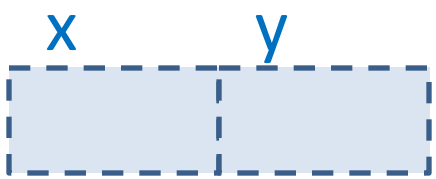
```
p1.x . . .  p1.y . . .
```

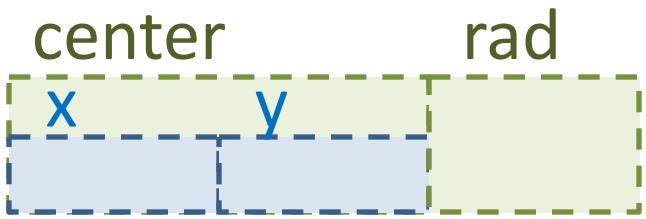


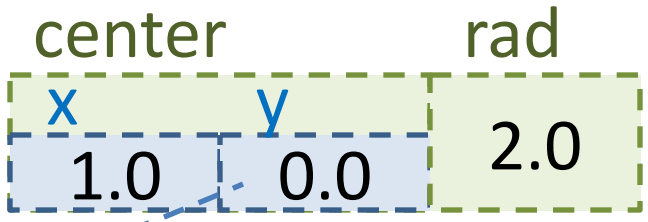
4.5. Вложенные структуры

```
struct  Point {
      double x,y;
};
struct  Circle {
      struct Point  center;
      double  rad;
};
 struct  Circle c = {{1, 0}, 2};

  c .center .y
```

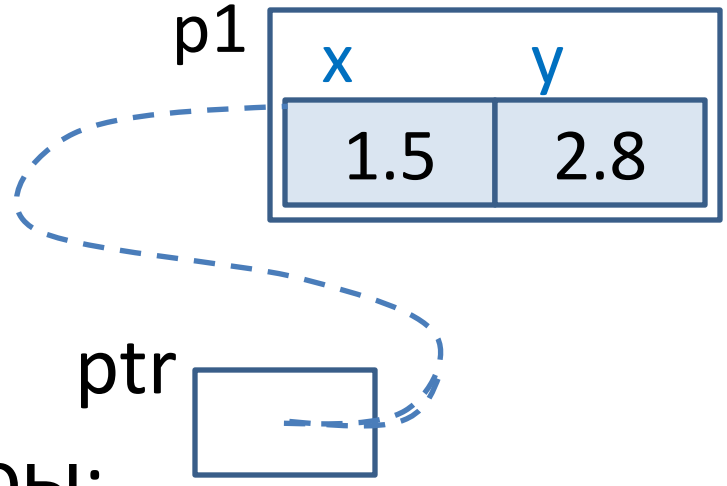






4.6. Указатели на структуру

```
struct  Point {
    double x, y;
};
struct  Point  p1 = {1.5, 2.8};
struct  Point  *ptr = &p1;
```



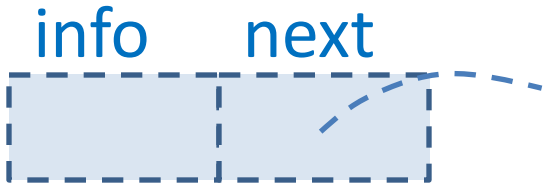
Разыменование структуры:

```
(*ptr).x        ptr->x
```

4.7. Списки

Элемент списка

```
struct  Item {
        тип  info;
        struct  Item *next;
};
```



Начало списка

```
struct  Item *first;
```

4.8. Примеры работы со списком

1. Создать список из символьной строки
2. Вывести список в выходной поток
3. Освободить память, занятую списком

4.8.1. Объявления

```
#include <stdio.h>
#include <malloc.h>

struct Item {
    char c;
    struct Item *next;
};

struct Item *creatList(const char *);
void putList(struct Item *);
struct Item *deleteList(struct Item *);
```

4.8.2. Функция createList()

```
struct Item *creatList(const char *str)
{
	struct Item head = {'*', NULL};
	struct Item *last = &head;
	while(*str != '\0'){
		last->next = (struct Item *)malloc(sizeof(struct Item));
		last = last->next;
		last->c = *str++;
		last->next = NULL;
	}
	return head.next;
}
```

4.8.3. Функция putList()

```
void putList(char *msg, struct Item *ptr)
{
    printf("%s: \"", msg);
    for(; ptr != NULL; ptr = ptr->next)
        printf("%c", ptr->c);
    printf("\"\n");
}
```

4.8.4. Функция deleteList()

```
struct Item *deleteList(struct Item *ptr)
{
    struct Item *tmp = NULL;
    while(ptr != NULL){
        tmp = ptr;
        ptr = ptr->next;
        free(tmp);
    }
    return ptr;
}
```

4.8.5. Тестирование

```
int main()
{
	char buf[80];
	struct Item *st;
	while(puts("enter string"), gets(buf)){
		st = creatList(buf);
		putList("Entered string", st);
		st = deleteList(st);
	}
	return 0;
}
```

4.9. Задача

Из входного потока вводится произвольное число текстовых строк; конец ввода – конец файла. Длина каждой строки также произвольна.

Каждая строка представляет собой последовательность слов, разделенных пробельными символами.

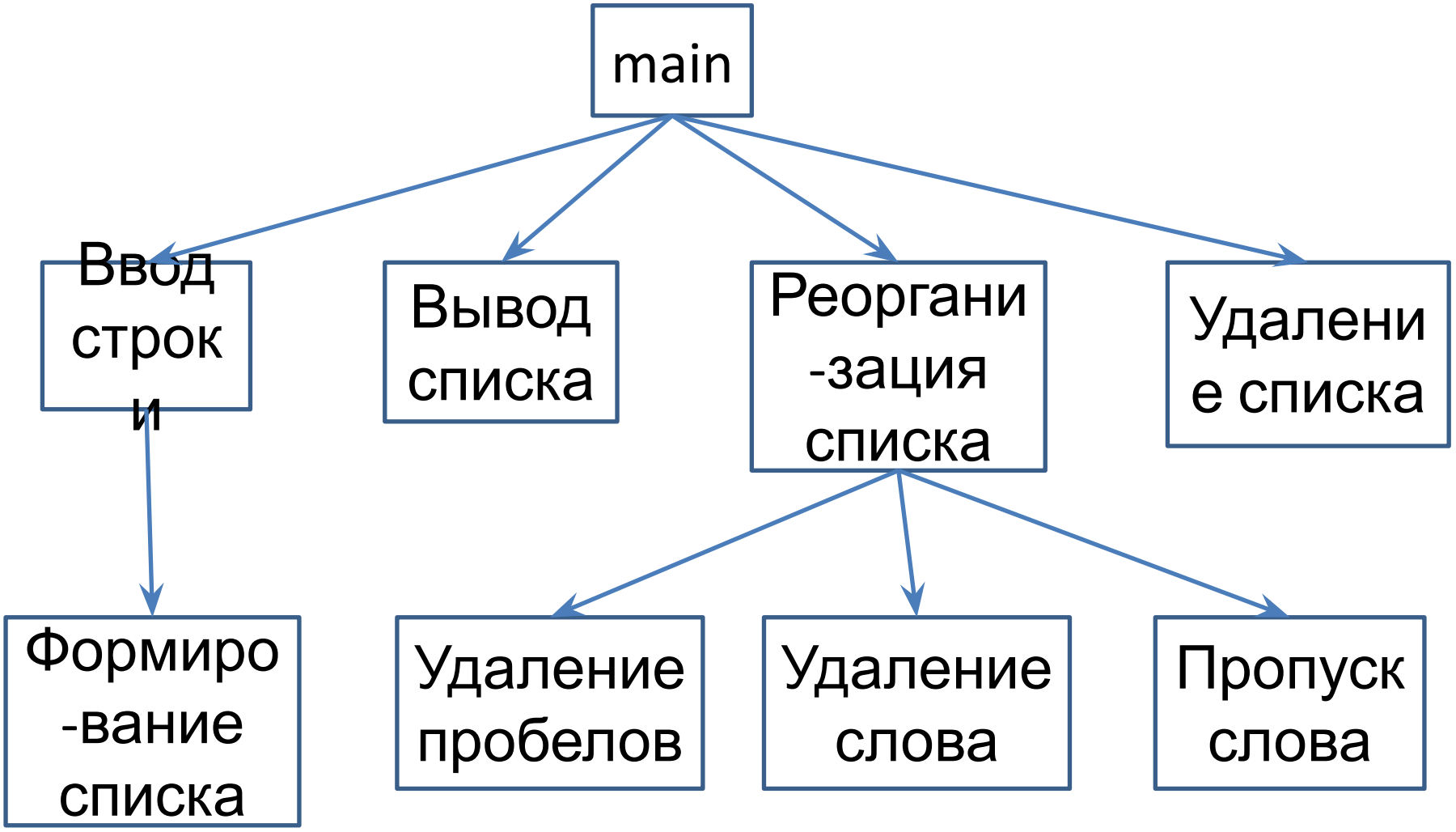
Получить новую строку, оставив в исходной ее каждое второе слово.

Строка представлена списком.

4.10. Структура программы

- Функция main()
- Функция ввода строки произвольной длины
- Функции создания списка, удаления списка и вывода списка в поток
- Функция формирования результирующей строки
- Функции удаления пробелов, удаления слова, пропуска слова

4.10. Структура программы



4.10.1. Функция main()

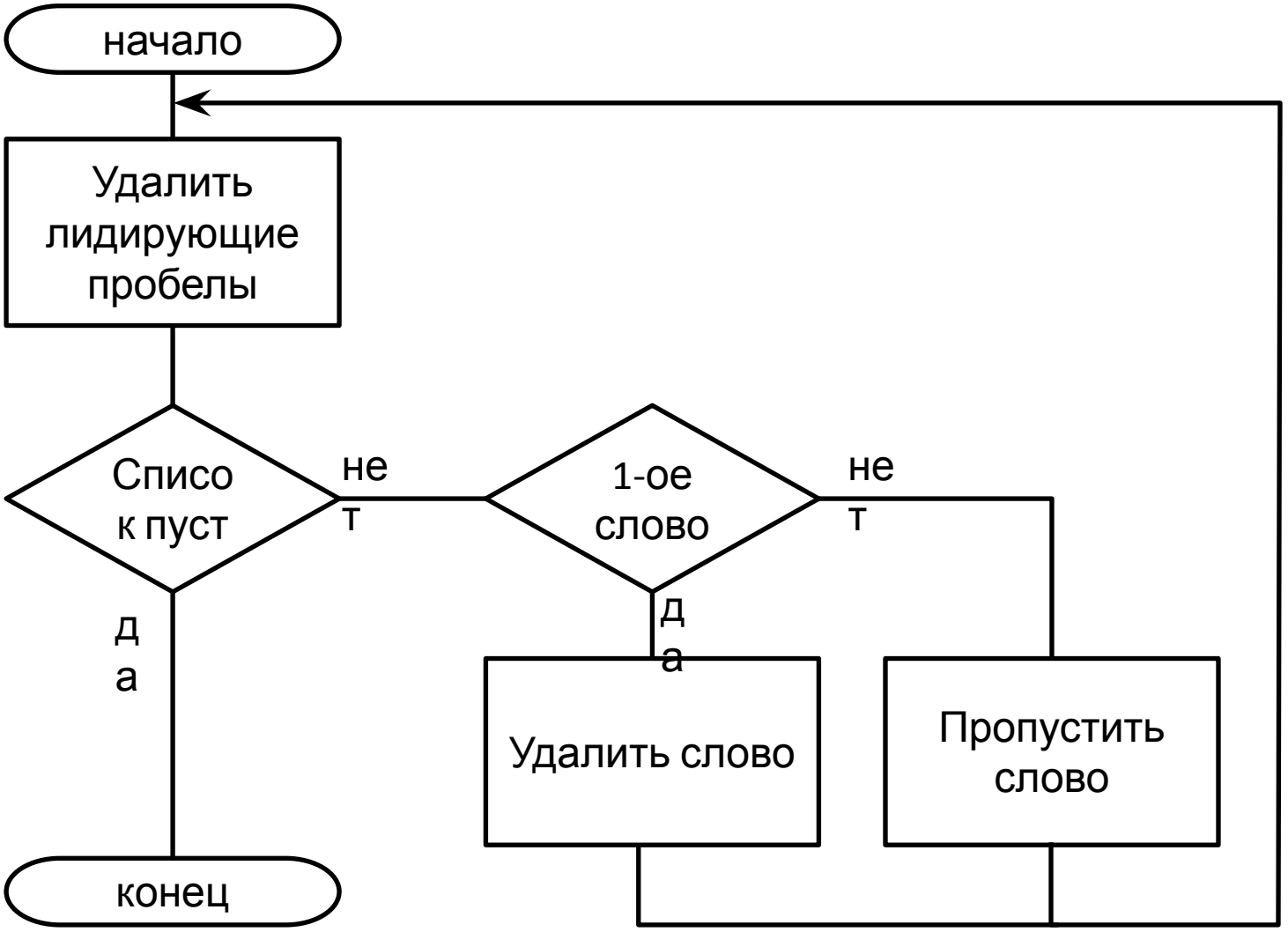
```
typedef struct Item{
   char c;
   struct Item *next;
} Item;

int main()
{
   Item *p = NULL;
```

4.10.2. Функция main()

```
while(puts("enter..."), getstr(&p) == 0){
    putList("Source string", p);
    p = reorg(p);
    putList("Result string", p);
    p = delList(p);
  }
  puts("That's all. Bye!");
  return 0;
}
```

4.11. Реорганизация списка



4.11.1. Реализация

```
Item *reorg(Item *p)
{
  Item head = {'\0', p},
    *last = &head,
    *prev = NULL;
  int f = 0;
  while(last && (last->next = delSpace(last->next))){
    f = !f;
```

4.11.2. Реализация

```
    if(f)
      last->next = delWord(last->next);
    else{
      prev = skipWord(last->next);
      last = prev->next;
      if(last)
        last->c = ' ';
    }  //  else
}  //  while
```

4.11.3. Реализация

```
    if(last && prev){
        prev->next = NULL;
        free(last);
    }
    return head.next;
}
```