Требования к программам

1. Программа должна получать все параметры в качестве аргументов командной строки.
2. Аргументы командной строки **для задач 1–2**:
   1) n – задает число повторений вызова функции и используется для измерения времени, имеет тип `size_t`,
   2) s_1 – строка.

   Например, запуск

   ```
   ./a01.out 10000000 "abcd"
   ```

   означает, что требуется измерить время 10000000 вызовов реализованной функции, которую требуется вызвать для строки `"abcd"`.
3. Аргументы командной строки **для задач 3–4**:
   1) n – задает число повторений вызова функции и используется для измерения времени, имеет тип `size_t`,
   2) s_1 – строка,
   3) ch – символ.

   Например, запуск

   ```
   ./a03.out 10000000 "abcd" 'b'
   ```

   означает, что требуется измерить время 10000000 вызовов реализованной функции, которую требуется вызвать для строки `"abcd"` и символа `'b'`.
4. Аргументы командной строки **для задач 5–10**:
   1) n – задает число повторений вызова функции и используется для измерения времени, имеет тип `size_t`,
   2) s_1 – строка,
   3) s_2 – строка.

   Например, запуск

   ```
   ./a05.out 10000000 "abcd" "efgh"
   ```

   означает, что требуется измерить время 10000000 вызовов реализованной функции, которую требуется вызвать для строк `"abcd"` и `"efgh"`.
5. Программа должна содержать функцию измерения времени работы n вызовов указанной функции. Функция тестирования получает в качестве аргументов n, указатель на тестируемую функцию, ее аргументы, указатель на место, куда надо записать результат тестируемой функции, и возвращает время работы в секундах. Прототипы функции тестирования
   - **для задачи 1**:

     ```
     double test_1 (size_t n, size_t (*p) (const char *),
                    const char *s, size_t *result);
     ```

- **для задач 2 и 5**:

```
double test_2_5 (size_t n, char* (*p) (char*, const char *),
                 char *s1, const char *s2, char **result);
```

в этих задачах после каждого вызова тестируемой функции надо приводить строку s_1 в исходное состояние.

- **для задач 3, 4**:

```
double test_3_4 (size_t n, char* (*p) (const char *, int),
                 const char *s, int ch, char **result);
```

- **для задачи 6**:

```
double test_6 (size_t n, int (*p) (const char*, const char *),
               const char *s1, const char *s2, int *result);
```

- **для задач 7, 8**:

```
double test_7_8 (size_t n, size_t (*p) (const char*, const char *),
                 const char *s1, const char *s2, size_t *result);
```

- **для задачи 9**:

```
double test_9 (size_t n, char* (*p) (const char*, const char *),
               char *s1, const char *s2, char **result);
```

- **для задачи 10**:

```
double test_10 (size_t n, char* (*p) (char*, const char *, char**),
                char *str, const char *delim, char **saveptr);
```

в этой задаче после каждого вызова тестируемой функции надо приводить строку *str* в исходное состояние.

6. Пример функции тестирования

```
double test_1 (size_t n, size_t (*p) (const char *),
               const char *s, size_t *result)
{
  double t; size_t i, r;
  t = clock ();
  for (i = 0; i < n; i++)
    r = (*p) (s);
  t = (clock () - t) / CLOCKS_PER_SEC;
  *result = r;
  return t;
}
```

7. Функция тестирования вызывается из основной программы `main` вначале для реализованной функции, и выводится результат ее работы (время и результат функции, указатель на которую передали). Затем функция тестирования вызывается для стандартной библиотечной функции, имеющей тот же прототип, но без символа подчеркивания `'_'` в конце, и выводится результат ее работы (время и результат функции, указатель на которую передали). Прототипы стандартных функций описаны в файле `<string.h>`.

8. Вывод результата работы (реализованной функции **и** соответствующей ей стандартной) в функции `main` **для задач 1, 7, 8** должен производиться по формату:

```
printf ("%s : Task = %d Res = %zd Elapsed = %.2f\n",
        argv[0], task, res, t);
```

где

- `argv[0]` – первый аргумент командной строки (имя образа программы),
- `task` – номер задачи,
- `res` – возвращаемое значение функции, реализующей решение этой задачи (типа `size_t`),
- `t` – время работы функции, реализующей решение этой задачи.

9. Вывод результата работы (реализованной функции **и** соответствующей ей стандартной) в функции `main` **для задачи 6** должен производиться по формату:

```
printf ("%s : Task = %d Res = %d Elapsed = %.2f\n",
        argv[0], task, res, t);
```

где `argv[0]`, `task`, `res`, `t` – означают то же, что в предыдущем пункте.

10. Вывод результата работы (реализованной функции **и** соответствующей ей стандартной) в функции `main` **для задач 2, 5** должен производиться по формату:

```
printf ("%s : Task = %d Res = %s Elapsed = %.2f\n",
        argv[0], task, res, t);
```

где `argv[0]`, `task`, `res`, `t` – означают то же, что в предыдущем пункте.

11. Вывод результата работы (реализованной функции **и** соответствующей ей стандартной) в функции `main` **для задач 3, 4, 9** должен производиться по формату:

```
printf ("%s : Task = %d Res = %s Elapsed = %.2f\n",
        argv[0], task, res, t);
```

где

- `argv[0]`, `task`, `t` – означают то же, что в предыдущем пункте,
- `res` равно возвращаемому значению функции, реализующей решение этой задачи, если оно не 0, или указателю на строку `"Not found"` иначе.

12. Вывод результата работы (реализованной функции **и** соответствующей ей стандартной) в функции `main` **для задачи 10** должен производиться по формату:

```
printf ("%s : Task = %d Res = %s Saveptr = %s Elapsed = %.2f\n",
        argv[0], task, res, next, t);
```

где

- `argv[0]`, `task`, `t` – означают то же, что в предыдущем пункте,
- `res` равно возвращаемому значению функции, реализующей решение этой задачи, если оно не 0, или указателю на строку `"Not found"` иначе,
- `next` равно значению указателя `saveptr` после работы функции, реализующей решение этой задачи, если оно не 0, или указателю на строку `"Not found"` иначе.

Задачи

1. Написать функцию

   ```
   size_t strlen_ (const char *string);
   ```

   которая возвращает длину в байтах строки, адрес которой определяется значением аргумента `string`, не учитывая завершающий нулевой символ.

2. Написать функцию

   ```
   char *strcpy_ (char *string1, const char *string2);
   ```

   которая копирует строку, адрес которой определяется значением аргумента `string2`, включая завершающий нулевой символ, по адресу, определяемому аргументом `string1`, и возвращает указатель на измененную строку (т.е. `string1`).

3. Написать функцию

   ```
   char *strchr_ (const char *string, int ch);
   ```

   которая возвращает указатель на первое вхождение символа, имеющего код `ch`, в строку, адрес которой определяется значением аргумента `string` (возвращается 0, если символ не найден). Завершающий нулевой символ включается в поиск.

4. Написать функцию

   ```
   char *strrchr_ (const char *string, int ch);
   ```

   которая возвращает указатель на последнее вхождение символа, имеющего код `ch`, в строку, адрес которой определяется значением аргумента `string` (возвращается 0, если символ не найден). Завершающий нулевой символ включается в поиск.

5. Написать функцию

   ```
   char *strcat_ (char *string1, const char *string2);
   ```

   которая добавляет строку, адрес которой определяется значением аргумента `string2`, включая завершающий нулевой символ, в конец строки, адрес которой определяется значением аргумента `string1`, и возвращает указатель на сцепленную строку (значение аргумента `string1`).

6. Написать функцию

   ```
   int strcmp_ (const char *string1, const char *string2);
   ```

   которая сравнивает строки, адреса которых задаются значениями аргументов `string1` и `string2`, лексикографически и возвращает значение, определяющее их соотношение:

   - <0, если `string1` лексикографически меньше, чем `string2`,
   - =0, если `string1` лексикографически равна `string2`,
   - >0, если `string1` лексикографически больше, чем `string2`.

7. Написать функцию

   ```
   size_t strcspn_ (const char *string1, const char *string2);
   ```

   которая возвращает индекс первого символа в строке, адрес которой определяется значением аргумента `string1`, который принадлежит набору символов, содержащихся в строке, адрес которой определяется значением аргумента `string2` (например, если строка `string1` не содержит ни одного символа из `string2`, то возвращается значение, равное длине `string1`, а если строка `string1` начинается с символа из `string2`, то возвращается 0).

8. Написать функцию

```
size_t strspn_ (const char *string1, const char *string2);
```

которая возвращает индекс первого символа в строке, адрес которой определяется значением аргумента `string1`, который не принадлежит набору символов, содержащихся в строке, адрес которой определяется значением аргумента `string2` (например, если строка `string1` содержит только символы из набора символов в строке `string2`, то возвращается значение, равное длине `string1`, а если строка `string1` начинается с символа не из набора символов в строке `string2`, то возвращается 0).

9. Написать функцию

```
char *strstr_ (const char *string1, const char *string2);
```

которая возвращает указатель на первое вхождение строки, адрес которой определяется значением аргумента `string2`, в строке, адрес которой определяется значением аргумента `string1` (возвращается 0, если вхождение не найдено).

10. Написать функцию

```
char *strtok_r_ (char *str, const char *delim, char **saveptr);
```

которая возвращает указатель на первое найденное слово в строке, адрес которой определяется значением аргумента `str`. Словом назывется последовательность символов, не содержащая в себе символов из строки разделитей, адрес которой определяется значением аргумента `delim`. Функция изменяет строку `str`, записывая нулевой символ на место первого найденного разделителя после конца слова. Адрес следующего символа после записанного нулевого сохраняется в `saveptr`. Функция возвращает 0, если слово не найдено.