

Лабораторная работа №11

по дисциплине «Программирование на Си»

Функции с переменным числом аргументов

Кострицкий А. С., Ломовской И. В.

Москва — 2020 — TS2011101155

Содержание

Цель работы	1
Задание	1
Взаимодействие с системой тестирования	2
Памятка преподавателя	4

Цель работы

Цель работы – приобрести навыки реализации функций с переменным числом параметров.

Студенты должны получить и закрепить на практике следующие знания и умения:

1. Работа с типом `va_list`.
2. Работа со строками.
3. Перевод чисел из одной системы счисления в другую.
4. Организация корректной работы с буфером ограниченного объема.

Задание

Реализовать собственную версию функции `snprintf`, обрабатывающую указанные спецификаторы типа. Функция должна называться `my_snprintf`. При реализации этой функции запрещается использовать любые стандартные функции для обработки строк.

Реализуемые спецификаторы (`%c`, `%d`, `%i`, `%x`, `%o`, `%s`) и модификаторы (`l` и `h`) распределяются преподавателем.

Исходный код лабораторной работы располагается в ветке `lab_11`. В этой ветке создается папка `lab_11`, в которой располагается исходный код программы. Реализованные спецификаторы сохраняются в файле `format.txt` в виде строки «`d hx s`» (кавычки не указываются). Для проверки правильности решения задачи реализуются только модульные тесты, в которых нужно сравнить поведение (спецификацию) своей функции и соответствующей стандартной.

Справка по функции (англ.):

<https://pubs.opengroup.org/onlinepubs/9699919799/functions/snprintf.html>

Взаимодействие с системой тестирования

1. Решение задачи оформляется студентом в виде многофайлового проекта. Для сборки проекта используется программа `make`, сценарий сборки `makefile` помещается под версионный контроль. В сценарии должны присутствовать цель `app.exe` — для сборки основной программы, и цель `unit_tests.exe` — для сборки модульных тестов.
2. В сценарии сборки рекомендуется обозначить, помимо прочих, следующие цели:
 - (a) `unit` — сборка и прогон модульных тестов.
 - (b) `func` — прогон функциональных тестов.
 - (c) `clean` — очистка генерируемых файлов.
3. Исходный код лабораторной работы размещается студентом в ветви `lab_LL`, а решение каждой из задач — в отдельной папке с названием вида `lab_LL_CC_PP`, где `LL` — номер лабораторной, `CC` — вариант студента, `PP` — номер задачи.

Пример: решения восьми задач седьмого варианта пятой лабораторной размещаются в папках `lab_05_07_01`, `lab_05_07_02`, `lab_05_07_03`, ..., `lab_05_07_08`.
4. Исходный код должен соответствовать оглащённым в начале семестра правилам оформления.
5. Если для решения задачи студентом создаётся отдельный проект в IDE, разрешается поместить под версионный контроль файлы проекта, добавив перед этим необходимые маски в список игнорирования. Старайтесь добавлять маски общего вида. Для каждого проекта должны быть созданы, как минимум, два варианта сборки: `Debug` — с отладочной информацией, и `Release` — без отладочной информации.
6. Для каждой программы ещё до реализации студентом заготавливаются и помещаются под версионный контроль в подпапку `func_tests` функциональные тесты, демонстрирующие её работоспособность.

Позитивные входные данные следует располагать в файлах вида `pos_TT_in.txt`, выходные — в файлах вида `pos_TT_out.txt`, аргументы командной строки при наличии — в файлах вида `pos_TT_args.txt`, где `TT` — номер тестового случая.

Негативные входные данные следует располагать в файлах вида `neg_TT_in.txt`, выходные — в файлах вида `neg_TT_out.txt`, аргументы командной строки при наличии — в файлах вида `neg_TT_args.txt`, где `TT` — номер тестового случая.

Разрешается помещать под версионный контроль в подпапку `func_tests` сценарии автоматического прогона функциональных тестов. Если Вы используете при автоматическом прогоне функциональных тестов сравнение строк, не забудьте проверить используемые кодировки. Помните, что UTF-8 и UTF-8(BOM) — две разные кодировки.

Под версионный контроль в подпапку `func_tests` также помещается файл `readme.md` с описанием в свободной форме содержимого каждого из тестов. Вёрстка файла на языке Markdown обязательной не является, достаточно обычного текста.

Пример: восемь позитивных и шесть негативных функциональных тестов без дополнительных ключей командной строки должны размещаться в файлах `pos_01_in.txt`, `pos_01_out.txt`, ..., `neg_06_out.txt`. В файле `readme.md` при этом может содержаться следующая информация:

```

# Тесты для лабораторной работы №LL

## Входные данные
Целые a, b, c

## Выходные данные
Целые d, e

## Позитивные тесты:
- 01 - обычный тест;
- 02 - в качестве первого числа ноль;
...
- 08 - все три числа равны.

## Негативные тесты:
- 01 - вместо первого числа идёт буква;
- 02 - вместо второго числа идёт буква;
...
- 06 - вводятся слишком большие числа.

```

7. Рекомендуется задавать следующую структуру проекта:

- (a) Все файлы исходных кодов хранятся в подпапке `src`.
- (b) Все файлы заголовков хранятся в подпапке `inc`.
- (c) Для каждого модуля создаётся и помещается в подпапку `unit_tests` один файл с модульными тестами, имя которого повторяет имя модуля с префиксом «`check_`». Основная программа модульного тестирования носит название «`check_main.c`».
- (d) Функциональные тесты оформляются в соответствии с предыдущими пунктами.
- (e) Сценарий сборки и конечные приложения генерируются в корне проекта.
- (f) Все остальные генерируемые файлы, в том числе объектные файлы и файлы статистики `gsov`, создаются в подпапке `out`.

Пример: папка с проектом для лабораторной работы, состоящего из текста программы и двух модулей, будет иметь следующий вид:

```
/lab_LL_CC_PP/  
  app.exe  
  makefile  
  unit_tests.exe  
  /inc/  
    unit_a.h  
    unit_b.h  
  /out/  
    main.o  
    unit_a.o  
    unit_b.o  
  /src/  
    main.c  
    unit_a.c  
    unit_b.c  
  /func_tests/  
    ...  
  /unit_tests/  
    check_main.c  
    check_unit_a.c  
    check_unit_b.c
```

8. Для каждой подпрограммы должны быть подготовлены модульные тесты с помощью фреймворка check, которые демонстрируют её работоспособность.
9. Все динамические ресурсы, которые уже были Вами успешно запрошены, должны быть высвобождены к моменту выхода из программы. Для контроля можно использовать, например, программы Dr. Memory или valgrind.
10. Успешность ввода должна контролироваться. При первом неверном вводе программа должна прекращать работу с ненулевым кодом возврата.

Обратите внимание, что даже в этом случае все динамические ресурсы, которые уже были Вами успешно запрошены, должны быть высвобождены.

11. Вывод Вашей программы может содержать текстовые сообщения и числа. Тестовая система анализирует только числа в потоке вывода, поэтому они могут быть использованы только для вывода результатов — использовать числа в информационных сообщениях запрещено.

Пример: сообщение «**Input point 1:**» будет неверно воспринято тестовой системой, а сообщения «**Input point A:**» или «**Input first point:**» — правильно.

12. Если не указано обратное, числа двойной точности следует выводить, округляя до шестого знака после запятой.

Памятка преподавателя

1. *Только для ЛР№11.* Совпадение структур и типов данных у студента и в задании не проверяется тестовой системой.