CSD1100 Programming Assignment: Lab 12 (Assembler - Arrays)

Topics and references

- Inspecting memory in gdb
- Arrays
- Developing and using functions

Task

In this assignment you have to implement 4 functions that take address of a null-terminated string as the first and only parameter and return result of calculations using basic arithmetic operators and flow control instructions.

Complete explanation how to work with bytes in memory and control the flow by using instructions **call** and **ret** and so on **will be given in the class**.

You implementation must passes all given tests in order to get the full mark for the assignment.

Submission details

Please read the following details carefully and adhere to all requirements to avoid unnecessary deductions.

Source files

You have to submit your implementation of the functions in the source file `functions.asm`.

```
; ------------------------------------------------------------------
; File: functions.asm
; Project: CSD1100 Assignment 12
; Author: Vadim Surov, vsurov@digipen.edu
; Co-Author: Your name, email
;
; Compile: nasm -f elf64 -g -F dwarf functions.asm -o functions.o
; Link: gcc main.o functions.o -o main.o -lm
; Run: ./main 0
; Debug: gdb main
;  (gdb) b len
;  (gdb) run
;  0
;  ...
;  To see a string in memory by address in rdi use gdb's command:
;  (gdb) p/d (char[20]) *$rdi
;   ...
;
; Copyright: 2021, Digipen Institute of Technology, Singapore
;
; Note: All functions use at most 6 parameters
;      p1, p2, p3, p4, p5, p6
;      located in registers
;       rdi, rsi, rdx, rcx, r8, r9
;       accordingly.
; Note 2: Use instruction cmp byte [rdi+rcx], 0 to campare
;         a byte by adddress rdi*rci with 0
; ------------------------------------------------------------------
    section .text

    global len
    global countn
    global counta
    global counts

len:
; TODO: - Given a null-terminated string. First parameter
;         has address of the string.
;       - Find and return the length of the string.
;       - The string can be any length. When empty, return 0.
;       - It's not allowed to use any standard library function.

    ret    ; return rax;



countn:
; TODO: - Given a null-terminated string. First parameter
;         has address of the string.
;       - Find and return the number of digits in
;         the string.
;       - The string can be any length. When empty, return 0.
;       - It's not allowed to use any standard library function.
; Hints:- Digit '0' has ASCII code 48, '1' - 49, ... '9' - 57.
;       - You can use your function len.

    ret    ; return rax;



counta:
; TODO: - Given a null-terminated string. First parameter
;         has address of the string.
;       - Find and return the number of alphabet letters in
;         the string.
;       - The string can be any length. When empty, return 0.
;       - It's not allowed to use any standard library function.
; Hints:- Letters have ASCII codes 65-90 and 97-122.
;       - You can use your function len.

    ret    ; return rax

counts:
; TODO: - Given a null-terminated string. First parameter
;         has address of the string.
;       - Find and return the number of special characters in
;         the string.
;       - Special character is a character which is neither a digit
;         not a letter.
;       - The string can be any length. When empty, return 0.
;       - It's not allowed to use any standard library function.
; Hint: - You can use your functions len, countn, and counta.

    ret    ; return rax
```

Test cases are given in `main.c` file. Do not change it. It won't be submitted anyway.

```c
/*----------------------------------------------------------------
  File: main.c
  Project: CSD1100 Assignment 12
  Author: Vadim Surov, vsurov@digipen.edu

  Compile: gcc -c -Wall -Wextra -Werror main.c -o main.o
  Link: gcc main.o functions.o -o main.o -lm

  Copyright: 2021, Digipen Institute of Technology, Singapore
----------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>

// See the function description in functions.asm
int len(char*);
int countn(char*);
int counta(char*);
int counts(char*);

void test1();
void test2();
void test3();
void test4();
void test5();
void test6();
void test7();
void test8();
void test9();
void test10();

int main(int argc, char* argv[])
{

    void (*f[])() = { test1, test2, test3, test4, test5, test6, test7, test8,
test9, test10 };
    const int SIZE = sizeof(f) / sizeof(f[0]);
    int id = -1;

    if (argc == 2)
    {
        if (argv[1][0] == 'i')
        {
            printf("Enter the test number or 0 to run all tests:\n");
            scanf("%d", &id);
        }
        else
            id = atoi(argv[1]);
    }
    else
        scanf("%d", &id);
    if (id == 0)
        for (int i = 0; i < SIZE; ++i)
            f[i]();
    else if (0 < id && id <= SIZE)
        f[id - 1]();
    else
        printf("Test %d not found.\n", id);

    return 0;
}



void test1()
{
    char str[] = "";
    int actual = len(str); /* Count the string length */
    int expected = 0;

    if (actual == expected)
        printf("Test 1 : Pass\n");
    else
        printf("Test 1 : Failed (%d)\n", actual);
}

void test2()
{
    char str[] = "Hello World!";
    int actual = len(str); /* Count the string length */
    int expected = 12;

    if (actual == expected)
        printf("Test 2 : Pass\n");
    else
        printf("Test 2 : Failed (%d)\n", actual);
}

void test3()
{
    char str[] = "";
    int actual = countn(str); /* Count the number of digit letters */
    int expected = 0;

    if (actual == expected)
        printf("Test 3 : Pass\n");
    else
        printf("Test 3 : Failed (%d)\n", actual);
}

void test4()
{
    char str[] = "1 a 2 b 3";
    int actual = countn(str); /* Count the number of digit letters */
    int expected = 3;

    if (actual == expected)
        printf("Test 4 : Pass\n");
    else
        printf("Test 4 : Failed (%d)\n", actual);
}

void test5()
{
    char str[] = "";
    int actual = counta(str); /* Count the number of alphabet letters */
    int expected = 0;

    if (actual == expected)
        printf("Test 5 : Pass\n");
    else
        printf("Test 5 : Failed (%d)\n", actual);
}

void test6()
{
    char str[] = "1 a 2 b 3";
    int actual = counta(str); /* Count the number of alphabet letters */
    int expected = 2;

    if (actual == expected)
        printf("Test 6 : Pass\n");
    else
        printf("Test 6 : Failed (%d)\n", actual);
}

void test7()
{
    char str[] = "";
    int actual = counts(str); /* Count the number of special characters */
    int expected = 0;

    if (actual == expected)
        printf("Test 7 : Pass\n");
    else
        printf("Test 7 : Failed (%d)\n", actual);
}

void test8()
{
    char str[] = "1 a 2 b 3";
    int actual = counts(str); /* Count the number of special characters */
    int expected = 4;

    if (actual == expected)
        printf("Test 8 : Pass\n");
    else
        printf("Test 8 : Failed (%d)\n", actual);
}

void test9()
{
    char str[] = "1234567890abcdefghijklmnopqrstuvwxyz[];',. ]";
    int actual = counts(str); /* Count the number of special characters */
    int expected = 8;

    if (actual == expected)
        printf("Test 9 : Pass\n");
    else
        printf("Test 9 : Failed (%d)\n", actual);
}

void test10()
{
    char str[] = "ABCDEFGHKLMNOPQRSTUVWXYZ1234567890!@#$%^&*()_+";
    int actual = counts(str); /* Count the number of special characters */
    int expected = 12;

    if (actual == expected)
        printf("Test 10 : Pass\n");
    else
        printf("Test 10 : Failed (%d)\n", actual);
}
```

Compiling, executing, and testing

Run `make` with the default rule to bring program executable `main` up to date:

```
$ make
```

Or, directly test your implementation by running `make` with target `test`:

```
$ make test
```

If the `diff` command in the `test` rule is not silent, then one or more of your function definitions is incorrect and will require further work.

File-level documentation

Every **edited by student** source file *must* begin with a *file-level* documentation block. This documentation serves the purpose of providing a reader the purpose of this source file at some later point of time. It has simple format. This module will not use any documentation generator like `Doxygen`.

```
; ------------------------------------------------------------------
; File: functions.asm
; Project: CSD1100 Assignment 12
; Author: Vadim Surov, vsurov@digipen.edu
; Co-Author: Your name, email
; ...
; ------------------------------------------------------------------
```

Submission and automatic evaluation

1. In the course web page, click on the appropriate submission page to submit `functions.asm`.
2. Please read the following rubrics to maximize your grade. Your submission will receive:
    - `F` grade if your `functions.asm` doesn't compile with the given options.
    - `F` grade if your `functions.asm` doesn't link to create an executable.

- Your implementation's output doesn't match correct output of the grader (you can see the inputs and outputs of the auto grader's tests). The auto grader will provide a proportional grade based on how many incorrect results were generated by your submission. `A+` grade if output of function matches correct output of auto grader.
- A deduction of one letter grade for each incorrect, incomplete or missing documentation block in `functions.asm`. For example, if the automatic grader gave your submission an `A+` grade and one documentation block is missing, your grade will be later reduced from `A+` to `B+`.