Week 07 Weekly Test Sample Answers

Test Conditions

These questions must be completed under self-administered exam-like conditions. You must time the test yourself and ensure you comply with the conditions below.

- You may complete this test in CSE labs or elsewhere using your own machine.
- You may complete this test at any time before **Wed Nov 04 21:00:00 2020**.
- Weekly tests are designed to act like a past paper - to give you an idea of how well you are progressing in the course, and what you need to work on. Many of the questions in weekly tests are from past final exams.
- Once the first hour has finished, you must submit all questions you've worked on.
- You should then take note of how far you got, which parts you didn't understand.
- You may choose then to keep working and submit test question anytime up to Wed Nov 04 21:00:00 2020
- However the maximum mark for any question you submit after the first hour will be 50%

You may access this **language documentation** while attempting this test:

- C quick reference
- MIPS quick reference

You may also access manual entries (the `man` command).

Any violation of the test conditions will results in a mark of zero for the entire weekly test component.

Set up for the test by creating a new directory called `test07`, changing to this directory, and fetching the provided code by running these commands:

```
$ mkdir test07
$ cd test07
$ 1521 fetch test07
```

Or, if you're not working on CSE, you can download the provided code as a zip file or a tar file.

WEEKLY TEST QUESTION:

MIPS - Which char?

You have been given `line_char.s`, a MIPS assembler program that reads a line of input then reads an integer, *n*, then prints a '?'.

```
$ 1521 spim -f line_char.s
Enter a line of input: Hello
Enter a position: 1
Character is: ?
```

Add code to `line_char.s` to make it equivalent to this C program which reads a line and prints the character in position *n*:

```c
//  read a line from stdin and and then an integer n
//  Print the character in the nth-position

#include <stdio.h>

// line of input stored here
char line[256];

int main(void) {

    printf("Enter a line of input: ");
    fgets(line, 256, stdin);

    printf("Enter a position: ");
    int n;
    scanf("%d", &n);

    printf("Character is: ");
    printf("%c", line[n]);
    printf("%c", '\n');

    return 0;
}
```

In other words change the code you given from printing a '?' to printing the first character of the line that has been read.

For example:

```
$ 1521 spim -f line_char.s
Enter a line of input: abcdefghijklmnopqrstuvwxyz
Enter a position: 0
Character is: a
$ 1521 spim -f line_char.s
Enter a line of input: abcdefghijklmnopqrstuvwxyz
Enter a position: 13
Character is: n
$ 1521 spim -f line_char.s
Enter a line of input: Hello Andrew!
Enter a position: 12
Character is: !
$ 1521 spim -f line_char.s
Enter a line of input: i MIPS you
Enter a position: 3
Character is: I
```

HINT:

Only a few instructions are needed.

NOTE:

You can assume a line can be read.

You can assume the line will contain at most 256 characters.

You can assume a positive integer **n** can be read.

You can assume the line read has at least **n + 1** characters.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 1521 autotest line_char
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1521 test07_line_char line_char.s
```

Sample solution for `line_char.s`

```
#  read a line from stdin and and then an integer n
#  Print the character in the nth-position

# n in $t0
# $t1, $t2 used to hold temporary values

main:
    la   $a0, str0       # printf("Enter a line of input: ");
    li   $v0, 4
    syscall

    la   $a0, line       # fgets(buffer, 256, stdin)
    la   $a1, 256
    li   $v0, 8
    syscall

    la   $a0, str1       # printf("Enter a position: ");
    li   $v0, 4
    syscall

    li   $v0, 5          # scanf("%d");
    syscall
    move $t0, $v0

    la   $a0, str2       # printf("Character is: ");
    li   $v0, 4
    syscall


    la   $t1, line       # printf("%c", line[n]);
    add  $t2, $t1, $t0
    lb   $a0, ($t2)
    li   $v0, 11
    syscall


    li   $a0, '\n'       # printf("%c", '\n');
    li   $v0, 11
    syscall

    li   $v0, 0          # return 0
    jr   $ra

.data
str0:
    .asciiz "Enter a line of input: "
str1:
    .asciiz "Enter a position: "
str2:
    .asciiz "Character is: "


# line of input stored here
line:
    .space 256
```

WEEKLY TEST QUESTION:

MIPS -How Long?

You have been given line_length.s, a MIPS assembler program that reads a line of input and then prints 42.

```
$ 1521 spim -f line_length.s
Enter a line of input: Hello
Line length: 42
```

Add code to `line_length.s` to make it equivalent to this C program which reads a line and prints its length:

```c
// read a line and print its length
#include <stdio.h>

// line of input stored here
char line[256];

int main(void) {
    printf("Enter a line of input: ");
    fgets(line, 256, stdin);

    int i = 0;
    while (line[i] != 0) {
        i++;
    }

    printf("Line length: ");
    printf("%d", i);

    printf("%c", '\n');
    return 0;
}
```

In other words change the code you given from printing 42 to printing the line length

For example:

```
$ 1521 spim -f line_length.s
Enter a line of input: Hello
Line length: 6
$ 1521 spim -f line_length.s
Enter a line of input: i MIPS you
Line length: 11
$ 1521 spim -f line_length.s
Enter a line of input: good bye
Line length: 9
```

NOTE:

As you can see in the above examples the characters read (and counted) include the '\n' at the end of the line.
You can assume a line containing at least 1 character is always successfully read.

No error handling is necessary.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 1521 autotest line_length
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1521 test07_line_length line_length.s
```

Sample solution for `line_length.s`

```
# read a line and print its length
# i in $t0
# $t1, $t2, $t3 used for temporary values

main:
    la   $a0, str0       # printf("Enter a line of input: ");
    li   $v0, 4
    syscall

    la   $a0, line
    la   $a1, 256
    li   $v0, 8          # fgets(buffer, 256, stdin)
    syscall              #

    li   $t0, 0          # i = 0;
loop:                    # while (line[i] != 0) {
    la   $t1, line
    add  $t2, $t1, $t0
    lb   $t3, ($t2)
    beq  $t3, 0, end
    addi $t0, $t0, 1     #   i++;
    j    loop            # }

end:
    la   $a0, str1       # printf("Line length: ");
    li   $v0, 4
    syscall

    move $a0, $t0        # printf("%d", i);
    li   $v0, 1
    syscall

    li   $a0, '\n'       # printf("%c", '\n');
    li   $v0, 11
    syscall

    li   $v0, 0          # return 0
    jr   $ra


.data
str0:
    .asciiz "Enter a line of input: "
str1:
    .asciiz "Line length: "


# line of input stored here
line:
    .space 256
```

WEEKLY TEST QUESTION:

MIPS - Palindromic?

You have been given palindrome.s, a MIPS assembler program that reads a line of input and then prints 2 lines.

```
$ 1521 spim -f palindrome.s
Enter a line of input: Hello
not palindrome
palindrome
```

Add code to `palindrome.s` to make it equivalent to this C program which reads a line and prints whether it is a palindrome or not.

```c
// read a line and print whether it is a palindrom
#include <stdio.h>

// line of input stored here
char line[256];

int main(void) {
    printf("Enter a line of input: ");
    fgets(line, 256, stdin);

    int i = 0;
    while (line[i] != 0) {
        i++;
    }
    int j = 0;
    int k = i - 2;
    while (j < k) {
        if (line[j] != line[k]) {
            printf("not palindrome\n");
            return 0;
        }
        j++;
        k--;
    }
    printf("palindrome\n");
    return 0;
}
```

A palindrome is a sequence which is the same forwards as backwards.

For example:

```
$ 1521 spim -f palindrome.s
Enter a line of input: kayak
palindrome
$ 1521 spim -f palindrome.s
Enter a line of input: canoe
not palindrome
$ 1521 spim -f palindrome.s
Enter a line of input: madamimadam
palindrome
$ 1521 spim -f palindrome.s
Enter a line of input: abcdecba
not palindrome
```

NOTE:

You can assume a line can be read.

You can assume the line will contain at most 256 characters.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 1521 autotest palindrome
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1521 test07_palindrome palindrome.s
```

Sample solution for `palindrome.s`

```
# read a line and print whether it is a palindrom

# i in $t0, j in $t1, k in $t2

# $t3, $t4, $t5, $t6, $t7 used for temporary values

main:
    la   $a0, str0       # printf("Enter a line of input: ");
    li   $v0, 4
    syscall

    la   $a0, line
    la   $a1, 256
    li   $v0, 8          # fgets(buffer, 256, stdin)
    syscall              #

    li   $t0, 0          # i = 0;
loop1:                   # while (line[i] != 0) {
    la   $t3, line
    add  $t4, $t3, $t0
    lb   $t5, ($t4)
    beq  $t5, 0, end1
    addi $t0, $t0, 1     #   i++;
    j    loop1           # }
end1:

    li   $t1, 0          # j = 0;
    addi $t2, $t0, -2    # k = i - 2;
loop2:                   # while (j < k) {
    bge  $t1, $t2, end2

    la   $t3, line
    add  $t4, $t3, $t1
    lb   $t5, ($t4)
    add  $t6, $t3, $t2
    lb   $t7, ($t6)
    beq  $t5, $t7, equal # if (line[j] != line[k]) {
    la   $a0, not_palindrome
    j    print_and_return
equal:
    addi $t1, $t1, 1     # j++;
    addi $t2, $t2, -1    # j--;
    j    loop2
end2:

    la   $a0, palindrome
print_and_return:
    li   $v0, 4
    syscall

    li   $v0, 0          # return 0
    jr   $ra


.data
str0:
    .asciiz "Enter a line of input: "
palindrome:
    .asciiz "palindrome\n"
not_palindrome:
    .asciiz "not palindrome\n"


# line of input stored here
line:
    .space 256
```

Submission

When you are finished each exercise make sure you submit your work by running **give**.

You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Wed Nov 04 21:00:00 2020** to complete this test.

Automarking will be run by the lecturer several days after the submission deadline for the test, using test cases that you haven't seen: different to the test cases `autotest` runs for you.

(Hint: do your own testing as well as running `autotest`)

Test Marks

After automarking is run by the lecturer you can view it here the resulting mark will also be available via via give's web interface or by running this command on a CSE machine:

```
$ 1521 classrun -sturec
```

The test exercises for each week are worth in total 1 marks.

The best 6 of your 8 test marks for weeks 3-10 will be summed to give you a mark out of 9.
