# Week 06 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 **Thu Oct 29 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 Thu Oct 29 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 `test06`, changing to this directory, and fetching the provided code by running these commands:

```
$ mkdir test06
$ cd test06
$ 1521 fetch test06
```

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 Are Not Negative

You have been given `not_negative.s`, a MIPS assembler program that reads a numbers and then prints it.

```
$ 1521 spim -f not_negative.s
Enter a number: 42
You entered: 42
```

Add code to `not_negative.s` to make it equivalent to this C program:

```c
// read numbers until a non-negative number entered
#include <stdio.h>

int main(void) {
    int x;

    while (1) {
        printf("Enter a number: ");

        scanf("%d", &x);

        if (x < 0) {
            printf("Enter a positive number\n");
        } else {
            printf("You entered: %d\n", x);
            break;
        }
    }

    return 0;
}
```

In other words it should read numbers until a non-negative number is entered,

For example:

```
$ 1521 spim -f not_negative.s
Enter a number: -5
Enter a positive number
Enter a number: -1
Enter a positive number
Enter a number: 24
You entered: 24
$ 1521 spim -f not_negative.s
Enter a number: -1
Enter a positive number
Enter a number: -2
Enter a positive number
Enter a number: -3
Enter a positive number
Enter a number: -4
Enter a positive number
Enter a number: -5
Enter a positive number
Enter a number: 0
You entered: 0
$ 1521 spim -f not_negative.s
Enter a number: 100
You entered: 100
```

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

```
$ 1521 autotest not_negative
```

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

```
$ give cs1521 test06_not_negative not_negative.s
```

Sample solution for `not_negative.s`

```
#  read numbers until a non-negative number entered
# x in $t0
main:
    la   $a0, str0       # printf("Enter a number: ");
    li   $v0, 4
    syscall

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

    bge  $t0, 0, positive
    la   $a0, str2       # printf("Enter a positive number\n");
    li   $v0, 4
    syscall
    j    main

positive:
    la   $a0, str1       # printf("You entered: ");
    li   $v0, 4
    syscall
    move $a0, $t0        # printf("%d", x);
    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 number: "
str1:
    .asciiz "You entered: "
str2:
    .asciiz "Enter a positive number\n"
```

WEEKLY TEST QUESTION:

## Reversed MIPS

You have been given reverse_negative.s, a MIPS assembler program that reads numbers into an array.

Add code to `reverse_negative.s` to make it equivalent to this C program:

```c
// Read numbers into an array until a negative number is entered
// then print the numbers in reverse order

#include <stdio.h>

int numbers[1000];

int main(void) {
    int i = 0;
    while (i < 1000) {
        int x;
        scanf("%d", &x);
        if (x < 0) {
            break;
        } else {
            numbers[i] = x;
        }
        i++;
    }

    while (i > 0) {
        i--;
        printf("%d\n", numbers[i]);
    }
}
```

In other words make it stop when a negative number is read and then print the numbers readin reverse order.

For example:

```
$ 1521 spim -f reverse_negative.s
1
2
3
4
-1
4
3
2
1
$ 1521 spim -f reverse_negative.s
15
4
1
42
-3
42
1
4
15
```

**NOTE:**

You can assume that a negative number will always be entered and at most 999 numbers are entered before a negative number is entered.

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

```
$ 1521 autotest reverse_negative
```

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

```
$ give cs1521 test06_reverse_negative reverse_negative.s
```

Sample solution for `reverse_negative.s`

```
# Read numbers into an array until a negative number is entered
# then print the numbers in reverse order

# i in register $t0
# registers $t1, $t2 & $t3 used to hold temporary results

main:
    li   $t0, 0         # i = 0
loop0:
    bge  $t0, 1000, end0# while (i < 1000) {

    li   $v0, 5         #   scanf("%d", &numbers[i]);
    syscall             #

    blt  $v0, 0, end0   # if (x < 0) break
    mul  $t1, $t0, 4    #   calculate &numbers[i]
    la   $t2, numbers   #
    add  $t3, $t1, $t2  #
    sw   $v0, ($t3)     #   store entered number in array

    addi $t0, $t0, 1    #   i++;
    j    loop0          # }
end0:

loop1:
    ble  $t0, 0, end1  # while (i > 0) {

    addi $t0, $t0, -1   #   i--

    mul  $t1, $t0, 4    #   calculate &numbers[i]
    la   $t2, numbers   #
    add  $t3, $t1, $t2  #
    lw   $a0, ($t3)     #   load numbers[i] into $a0

    li   $v0, 1         #   printf("%d", numbers[i])
    syscall

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

    j    loop1          # }
end1:

    li   $v0, 0         # return 0
    jr   $ra

.data
numbers:
    .space 4000
```

WEEKLY TEST QUESTION:

## Different MIPS

You have been given different10.s, a MIPS assembler program that reads 10 numbers into an array. Add code to `different10.s` to make it equivalent to this C program:

```c
#include <stdio.h>

int numbers[10];

int main(void) {
    int x, i, n_seen;

    n_seen = 0;
    while (n_seen < 10) {
        printf("Enter number: ");
        scanf("%d", &x);

        i = 0;
        while (i < n_seen) {
            if (x == numbers[i]) {
                break;
            }
            i++;
        }

        if (i == n_seen) {
            numbers[n_seen] = x;
            n_seen++;
        }
    }
    printf("10th different number was %d\n", x);

    return 0;
}
```

In other words make it stop when 10 different numbers have been read and print a message including

For example:

```
$ 1521 spim -f different10.s
Enter a number: 11
Enter a number: 12
Enter a number: 13
Enter a number: 14
Enter a number: 15
Enter a number: 16
Enter a number: 17
Enter a number: 18
Enter a number: 19
Enter a number: 20
10th different number was: 20
$ 1521 spim -f different10.s
Enter a number: 11
Enter a number: 11
Enter a number: 12
Enter a number: 11
Enter a number: 13
Enter a number: 14
Enter a number: 13
Enter a number: 14
Enter a number: 15
Enter a number: 16
Enter a number: 29
Enter a number: 19
Enter a number: 18
Enter a number: 55
10th different number was: 55
```

**NOTE:**

You can assume that a 10 different numbers will always be entered.

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

```
$ 1521 autotest different10
```

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

```
$ give cs1521 test06_different10 different10.s
```

Sample solution for `different10.s`

```
# x in register $t0
# i in register $t1
# n_seen in register $t2
# registers $t3 and $t4 used to hold temporary results
main:
    li   $t2, 0         # n_seen = 0;
start:
    bge  $t2, 10, end   # while (n_seen < 10) {
    la   $a0, string0   # printf("Enter a number: ");
    li   $v0, 4
    syscall
    li   $v0, 5         # scanf("%d", &x);
    syscall
    move $t0, $v0
    li   $t1, 0         # i = 0
loop:
    bge  $t1, $t2, done # while (i < n_seen) {
    mul  $t3, $t1, 4    # printf("%d", numbers[count])
    la   $t4, numbers   # calculate &numbers[i]
    add  $t3, $t3, $t4  #
    lw   $t4, ($t3)     # load numbers[i] into $t4

    beq  $t4, $t0, start# if (x == numbers[i])
    addi $t1, $t1, 1    # i++;
    j    loop
done:
    bne  $t1, $t2, loop # if (i == n_seen)
    mul  $t3, $t2, 4    #
    la   $t4, numbers   # calculate &numbers[n_seen]
    add  $t3, $t3, $t4  #
    sw   $t0, ($t3)     # numbers[n_seen] = x
    addi $t2, $t2, 1    # n_seen++;
    j    start


end:
    la   $a0, string1   # printf("10th different number was: ");
    li   $v0, 4
    syscall

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

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

    li   $v0, 0         # return 0
    jr   $ra

    .data

numbers:
    .space 40           # int numbers[10];

string0:
    .asciiz "Enter a number: "
string1:
    .asciiz "10th different number was: "
```

## 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

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 **Thu Oct 29 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.