Week 09 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 18 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 18 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 `test09`, changing to this directory, and fetching the provided code by running these commands:

```
$ mkdir test09
$ cd test09
$ 1521 fetch test09
```

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:

Create An add Instruction

Your task is to add code to this function in **add.c**:

```
// return the MIPS opcode for add $d, $s, $t
uint32_t add(uint32_t d, uint32_t s, uint32_t t) {

    return 42; // REPLACE WITH YOUR CODE

}
```

The function `add` is given the operands for a MIPS **add** instruction . Add code so that it returns the opcode for that instruction. Reminder the bit pattern for MIPS add instruction is:

Assembler	Description	C	Bit Pattern
`add $d, $s, $t`	add	d = s + t	`000000ssssstttttddddd00000100000`

This is how your code should behave

```
$ ./add 17 19 3
add(17, 19, 3) returned  0x02638820
$ ./add 9 27 12
add(9, 27, 12) returned 0x036c4820
```

Use *make* to build your code:

```
$ make    # or 'make add'
```

Assumptions/Limitations/Clarifications

You may define and call your own functions if you wish.

You are not permitted to change the `main` function you have been given, or to change `add`' prototype (its return type and argument types).

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

```
$ 1521 autotest add
```

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

```
$ give cs1521 test09_add add.c
```

Sample solution for `add.c`

```c
// Sample solution for COMP1521 lab exercises
//
// generate the opcode for an add instruction

#include <stdio.h>
#include <stdint.h>
#include <assert.h>

#include "add.h"

// return the MIPS opcode for add $d, $s, $t

uint32_t add(uint32_t d, uint32_t s, uint32_t t) {
    assert(0 <= d && d < 32);
    assert(0 <= s && s < 32);
    assert(0 <= t && t < 32);
    return s << 21 |
           t << 16 |
           d << 11 |
           0x20;
}
```

WEEKLY TEST QUESTION:

Print A String

Your task is to add code to this function in **put_string.c**:

```c
void put_string(char *s) {

   // PUT YOUR CODE HERE

}
```

The function `put_string` is given a string. It should write the string to stdout followed by a newline. It should use the *fputc* function do this. You are not permitted to use any other function.

```
$ ./put_string hello
calling put_string("hello"):
hello
$ ./put_string 'good bye'
calling put_string("good bye"):
good bye
```

Use *make* to build your code:

```
$ make    # or 'make put_string'
```

Assumptions/Limitations/Clarifications

You may define and call your own functions if you wish.

You are not permitted to change the `main` function you have been given, or to change `put_string`' prototype (its return type and argument types).

You are not permitted to use printf, puts, fputs or write.

You are not permitted to use C library function other than *fputc*.

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

```
$ 1521 autotest put_string
```

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

```
$ give cs1521 test09_put_string put_string.c
```

Sample solution for `put_string.c`

```c
// Sample solution for COMP1521 lab exercises

#include <stdio.h>

#include "put_string.h"

// print s to stdout with a new line appended using fputc (only)

void put_string(char *s) {
    while (*s != '\0') {
        fputc(*s, stdout);
        s++;
    }
    fputc('\n', stdout);
}
```

WEEKLY TEST QUESTION:

Read A Line

Your task is to add code to this function in **get_string.c**:

```c
// print a line from stream using fgetc (only)
// reads  in at  most one less than size characters from stream and stores them into the
// buffer pointed to by s.  Reading stops after an EOF or a newline.  If a newline is read, it  is
// stored  into  the buffer.  A terminating null byte ('\0') is stored after the last character in the buffer.
void get_string(char *s, int size, FILE *stream) {

    // PUT YOUR CODE HERE

}
```

The function `get_string` should read a line of input from the specified stream.

More precisely it should: read in at most one less than size characters from stream and store them into the buffer pointed to by s. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A terminating null byte ('\0') is stored after the last character in the buffer

For example:

```
$ ./get_string 12
calling get_string(s, 12, stdin):
hello
s now contains 'hello
'
$ ./get_string 16
calling get_string(s, 16, stdin):
good bye
s now contains 'good bye
'
$ ./get_string 5
calling get_string(s, 5, stdin):
hello
s now contains 'hell'
```

Use *make* to build your code:

```
$ make    # or 'make get_string'
```

Assumptions/Limitations/Clarifications

You may define and call your own functions if you wish.

You are not permitted to change the `main` function you have been given, or to change `get_string`' prototype (its return type and argument types).

You are not permitted to use fscanf fgets, read.

You are not permitted to use C library functions other than *fgetc*.

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

```
$ 1521 autotest get_string
```

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

```
$ give cs1521 test09_get_string get_string.c
```

Sample solution for `get_string.c`

```c
// Sample solution for COMP1521 lab exercises

#include <stdio.h>

#include "get_string.h"

// print a line from stream using fgetc (only)
// reads  in at  most one less than size characters from stream and stores them into the
// buffer pointed to by s.  Reading stops after an EOF or a newline.  If a newline is read, it  is
// stored  into  the buffer.  A terminating null byte ('\0') is stored after the last character in the buffer.
void get_string(char *s, int size, FILE *stream) {

    int bytes_read = 0;
    int c;
    while (bytes_read < size - 1 && (c = fgetc(stream)) != EOF) {
        s[bytes_read] = c;
        bytes_read++;
        if (c == '\n') {
            break;
        }
    }
    s[bytes_read] = '\0';

}
```

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 18 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.