Programming Fundamentals Exam
December 2020

Time recommended for the exam: 3 hours
Time allowed for entire final exam: 6 hours
Total number of questions: 20 short answer + 8 practical questions
Total number of marks: 100
Due: Friday 4 December 18:00 AEDST
Answer all questions

Current Time in AEDST

12:09 AEDST

Paper Information

Name	Wang, Jinghan
zID	z5286124
Tut/Lab	fri15a

This paper is unique to you. Submitting questions from another paper may result in charges of academic misconduct, and a grade of 0 in this course.

You should keep this paper confidential. Sharing it, during or after the exam is prohibited. **You MUST NOT discuss the paper until 24 hours after it starts. Some students have extensions!**

Do not place your exam work in any location accessible to any other person. This includes services such as Dropbox and Github.

Ensure during the exam no other person in your household can access your work.

Your zpass should not be disclosed to any other person. If you have disclosed your zpass, you should change it immediately.

To enable motivational images, read to the end of the paper.

Commencing Exam

If you are unwell, or will not be able to complete this exam within 6 hours, do not commence this exam - contact an exam supervisor at cs1511.exam@cse.unsw.edu.au.

Exam Conditions

This exam is **Open Book**, but you may not confer with anyone else, or submit another person's code. You may not ask for help from online sources. This means that you may use any resources from the course, or from online. You may not confer with anybody (including other students).

If there are any issues during the exam, or you have questions, contact an exam supervisor at cs1511.exam@cse.unsw.edu.au.

Special Considerations

This exam is covered by the Fit-to-Sit policy. That means that by sitting this exam, you are declaring yourself well enough to do so. You will be unable to apply for special consideration after the exam for circumstances affecting you before it began. If you have questions, or you feel unable to complete the exam, contact cs1511.exam@cse.unsw.edu.au

If you experience a technical issue before or during the exam, you should follow the following instructions:

Take screenshots of as many of the following as possible:

- error messages
- screen not loading
- timestamped speed tests
- power outage maps
- messages or information from your internet provider regarding the issues experienced

You should then get in touch with us via cs1511.exam@cse.unsw.edu.au

Exam Hurdle Requirements

COMP1511 has two hurdle requirements on this part of the final exam which you must meet to pass the course.

You must satisfactorily answer a question below which says it meets the arrays hurdle requirement.

Multiple questions are marked as meeting the arrays hurdle requirement. Answering any one of these questions satisfactorily will meet the hurdle requirement.

You must satisfactorily answer a question below which says it meets the linked lists hurdle requirement.

Multiple questions are marked as meeting the linked lists hurdle requirement. Answering any one of these questions satisfactorily will meet the hurdle requirement.

Exam Environment

You may complete the exam questions anywhere you wish. You should ensure they work correctly, and pass autotests, on VLab, or over SSH.

Autotests are available for all questions to assist you in your testing.

Passing autotests does not guarantee any marks.

Some Questions may deliberately exclude important autotests. If a question says it does excludes some autotests, make sure you check it works correctly.

Submission

Answer each question in a **SEPARATE** file. Each question specifies the name of the file to use. These are named after the corresponding question number, i.e. Question 1 is in a file called **exam_q1.c**

Make sure you use **EXACTLY** this file name.

Submit these files using the **submit** command as described in each question. **You may submit your answers as many times as you like.** The last submission **ONLY** will be marked.

Ensure that you have submitted your files before signing out. Running the autotests does not automatically submit your code.

Language Restrictions

All questions must be answered entirely in C. You may not use other programming languages, e.g. you may not use Python.

Your C program can not run external programs, e.g. it can not call the function **system()**.

You are not permitted to use functions from C libraries other than the standard C libraries (stdio.h, stdlib.h, string.h, math.h, ctype.h, assert.h).

Individual questions may specify C features you are not permitted to use or C functions you are not permitted to use.

You can not use the -l option to dcc.

Setup

This exam comes with starter files. You can access them by running:

```
$ 1511 fetch-exam
```

This gives you a file in which to write answers to the short answer questions. This file, named `exam_mc.txt`, is the template into which you should put your responses to each multiple-choice question.

This also gives you starter files for each programming question.

To test your code, run the command:

```
$ 1511 autotest-exam exam_q1
```

Short Answer Questions (20 marks total)

> **Note:** The files mentioned in the following questions are not copied by `1511 fetch-autotest`. The file names are just to show how they were compiled.

Question 1 (1 mark)

Imagine a file `part1_q1.c` (which has not been provided) contains this C Code (amongst other code):

```
int x = 10;
int y = 3;
printf("%d", x % y);
```

Imagine `part1_q1.c` is compiled with dcc on a CSE machine like this:

```
$ dcc part1_q1.c -o part1_q1
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./part1_q1
```

What would this program print?

Your answer should be the output the code snippet produces.

Enter this answer in the file `exam_mc.txt`, after the `[q1]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Do not enter any extra characters. Do not write \n for a newline character. Do not enter any explanation.

Enter just the output the program produces.

If the program prints an error message just write **ERROR**. Do not enter the exact error message.

Question 2 (1 mark)

Imagine a file named `part1_q2.c` (which has not been provided) contains this C Code (amongst other code):

```
char name[] = "Mishka";
double age = 0.5;
double weight = 1.5;
if (age > 5) {
    printf("%s is an old cat who is %lf years old.", name, age);
} else if (weight > 2) {
    printf("%s is a chonker.", name);
} else {
    printf("%s is a young kitten.", name);
}
```

Imagine `part1_q2.c` is compiled with dcc on a CSE machine like this:

```
$ dcc part1_q2.c -o part1_q2
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./part1_q2
```

What does this program print?

Your answer should be the output the code snippet produces.

Enter this answer in the file `exam_mc.txt`, after the `[q2]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Do not enter any extra characters. Do not write \n for a newline character. Do not enter any explanation.

Enter just the output the program produces.

If the program prints an error message just write **ERROR**. Do not enter the exact error message.

Question 3 (1 mark)

Imagine a file named `part1_q3.c` (which has not been provided) contains this C Code (amongst other code):

```
void print_time(int hour, int minute, int military_time) {
    if (military_time != 1) {
        if (hour > 0 && hour < 13) {
            printf("%d:%d AM", (hour % 12), minute);
        } else {
            printf("%d:%d PM", (hour % 12), minute);
        }
    } else {
        printf("%d%d", hour, minute);
    }
}
```

Imagine `part1_q3.c` is compiled with dcc on a CSE machine like this:

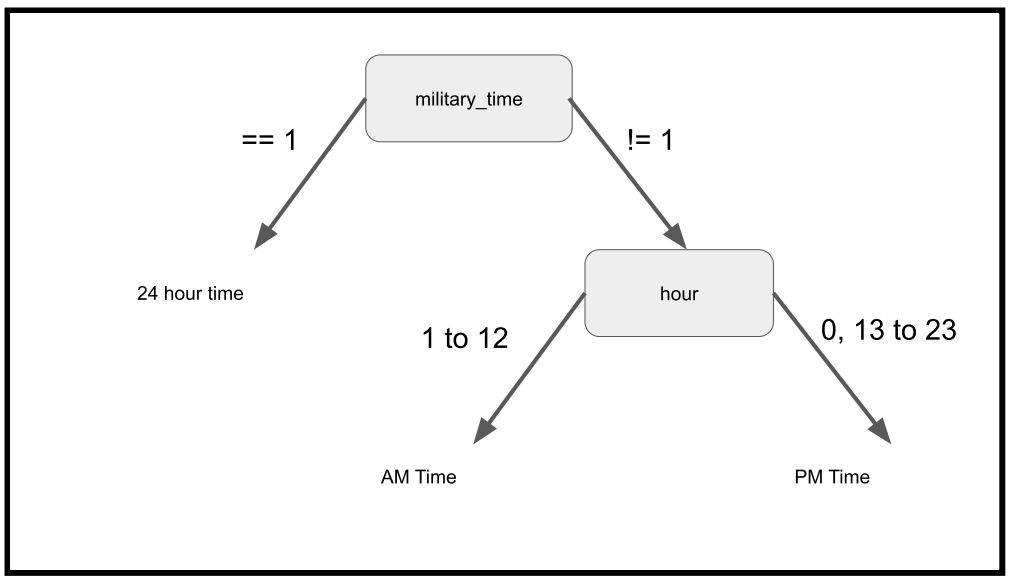
```
$ dcc part1_q3.c -o part1_q3
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

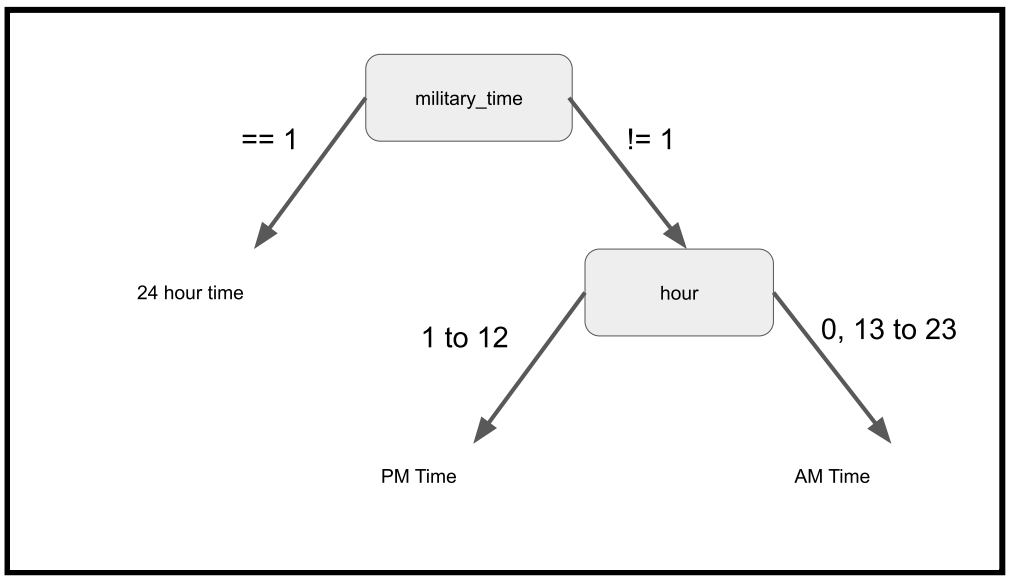
```
$ ./part1_q3
```

Which of the following diagrams best represents this if statement chain:

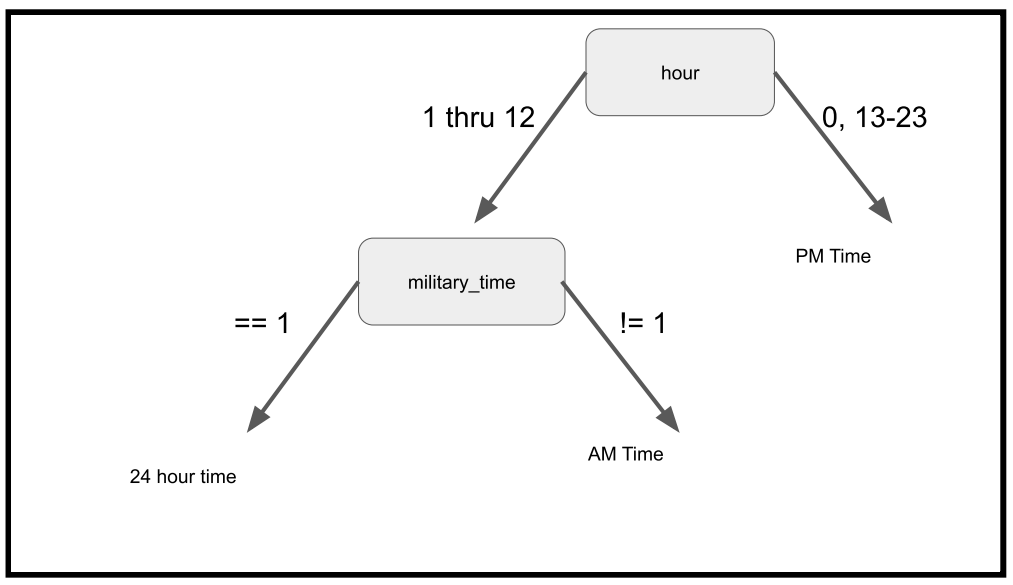
- **A**



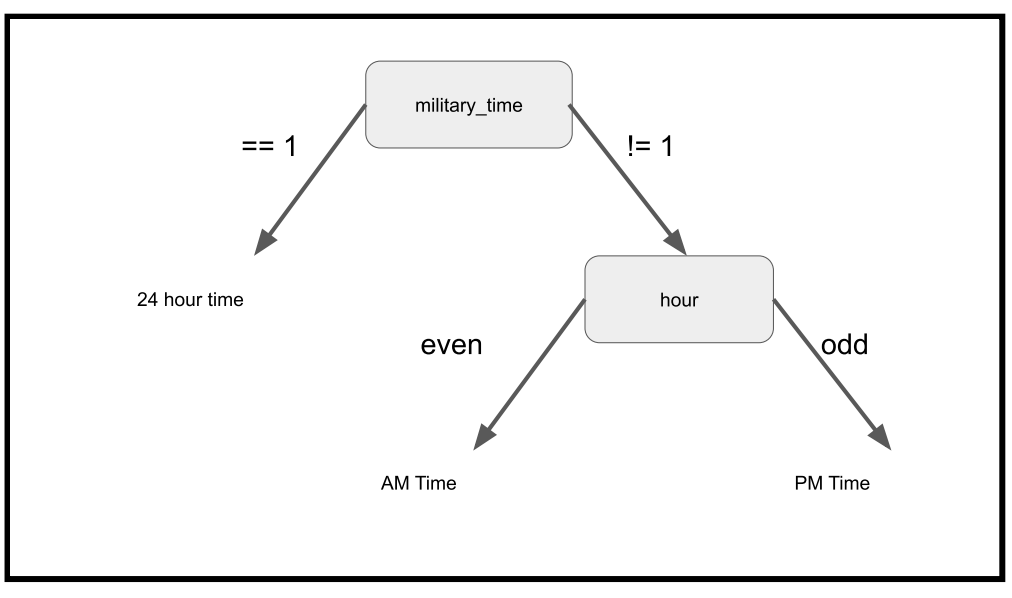
- **B**



- **C**



- **D**



Your answer should be either A, B, C, or D.

Enter this answer in the file `exam_mc.txt`, after the `[q3]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Do not enter any extra characters. Do not write \n for a newline character. Do not enter any explanation.

Enter just the letter A, B, C, or D.

Question 4 (1 mark)

Imagine a file named `part1_q4.c` (which has not been provided) contains this function:

```
void secret_function(char my_string[10]) {
    printf("%c %d", my_string[3], my_string[4]);
}
```

This program compiles successfully. When compiled, no errors or warnings are produced by dcc.

Give an example of an input that would cause this program to print `d 0`.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Do not enter any extra characters. Do not write \n for a newline character. Do not enter any explanation.

Enter just the output the program produces.

If the program prints an error message just write **ERROR**. Do not enter the exact error message.

Question 5 (1 mark)

Imagine a file named `part1_q5.c` (which has not been provided) contains this C Code (amongst other code):

```
int mystery(int a, int b, int c) {
    int answer = a;
    if (a > b) {
        b = a;
    }
    if (c > b) {
        answer = c;
    } else {
        answer = b;
    }
    return answer;
}

int main(void) {
     int w = 8;
     int x = 6;
     int y = 4;
     int z = 2;
     w = mystery(x, y, z);
     printf("w = %d", w);
     return 0;
}
```

Imagine `part1_q5.c` is compiled with dcc on a CSE machine like this:

```
$ dcc part1_q5.c -o part1_q5
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./part1_q5
```

What does this program print?

Your answer should be the output the code snippet produces.

Enter this answer in the file `exam_mc.txt`, after the `[q5]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Do not enter any extra characters. Do not write \n for a newline character. Do not enter any explanation.

Enter just the output the program produces.

If the program prints an error message just write **ERROR**. Do not enter the exact error message.

Question 6 (1 mark)

Imagine a file named `part1_q6.c` (which has not been provided) contains this C Code (amongst other code):

```c
#include <stdio.h>


void multiply([SECRET]) {
    *result_ptr = 5 * multiply_factor;
}

int main(void) {
    int result;
    int multiply_factor = 5;
    multiply(&result, multiply_factor);

    printf("%d", result);
}
```

The answer to this question is a piece of code that could replace **[SECRET]** on line 4.

For instance, if you believe that the correct text of line 4 is `void multiply(char a, char b) {`, you would write **char a, char b** as your answer.

Your answer should ensure that dcc compiles the program correctly; and that there will be no runtime errors.

You may not change any other lines of the program.

Question 7 (1 mark)

Imagine a file named `part1_q7.c` (which has not been provided) contains this C Code (amongst other code):

```c
#include <stdio.h>

struct example {
    int x,
    int y;
};

int main(void) {
    struct exmpl s;
    s.x = 0; s.y = 5;
    printf("%d", s.x);;
    return s.x;
```

Imagine `part1_q7.c` is compiled with dcc on a CSE machine like this:

```
$ dcc part1_q7.c -o part1_q7
```

It does not compile successfully.

What is the **first** line number that has code with an error?

Your answer should be a single integer between 1 and 12.

For example, if you believed there was an error on lines 5 and 7, you would write 5, since that is the first line that you think has an error.

Enter this answer in the file `exam_mc.txt`, after the `[q7]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Do not enter any extra characters. Do not write \n for a newline character. Do not enter any explanation.

Question 8 (1 mark)

Imagine a file named `part1_q8.c` (which has not been provided) contains this C Code (amongst other code):

```c
int x = 1;
int y = 0;
while (x < 9) {
    y = y + x;
    x = x + 3;
}
```

Imagine `part1_q8.c` is compiled with dcc on a CSE machine like this:

```
$ dcc part1_q8.c -o part1_q8
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./part1_q8
```

`y` has four different values during the running of this code.

Your answer should be the four separate integer values with spaces in between them.

Enter this answer in the file `exam_mc.txt`, after the `[q8]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Do not enter any extra characters. Do not write \n for a newline character. Do not enter any explanation.

For example, if you think that the values are 1, 2, 3 then 4 (in that order), your answer will be: 1 2 3 4

Question 9 (1 mark)

Imagine a file named `part1_q9.c` (which has not been provided) contains this C Code (amongst other code):

Note that part of a line has been removed, as you will have to fill it in.

```
int array[5] = {[SECRET]};
int i = 0;
int sum = 0;
while (i < 5) {
    sum = sum + array[i];
    printf("%d\n", sum);

    i++;
}
```

Imagine `part1_q9.c` is compiled with dcc on a CSE machine like this:

```
$ dcc part1_q9.c -o part1_q9
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./part1_q9
2
5
4
4
7
```

What is the piece of code that should replace **[SECRET]**?

Your answer should be a piece of code. If inserted into the file, your answer should ensure the program compiles correctly, and produces the above output.

Enter this answer in the file `exam_mc.txt`, after the `[q9]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Do not write \n for a newline character. Do not enter any explanation.

Question 10 (1 mark)

Imagine a file named `part1_q10.c` contains this C Code:

```
// fox bites @ UNSW.
#include <stdio.h>

int main(void) {
    int foxes = 6;
    int bites = 0;
while (foxes <= 14) {
    while (bites < 8) {


        printf("%d foxes bit students %d times.\n", foxes, bites);
        bites++;
    }
    foxes++;
}
}
```

Which of the following Code Style Issues are present in this code?

- A: Lack of use of #define constants for important values
- B: Overly long functions
- C: Inconsistent Indentation
- D: Overly Deep Nesting
- E: Bad Header Comment.

Enter the answer(s) in the file `exam_mc.txt`, after the `[q10]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

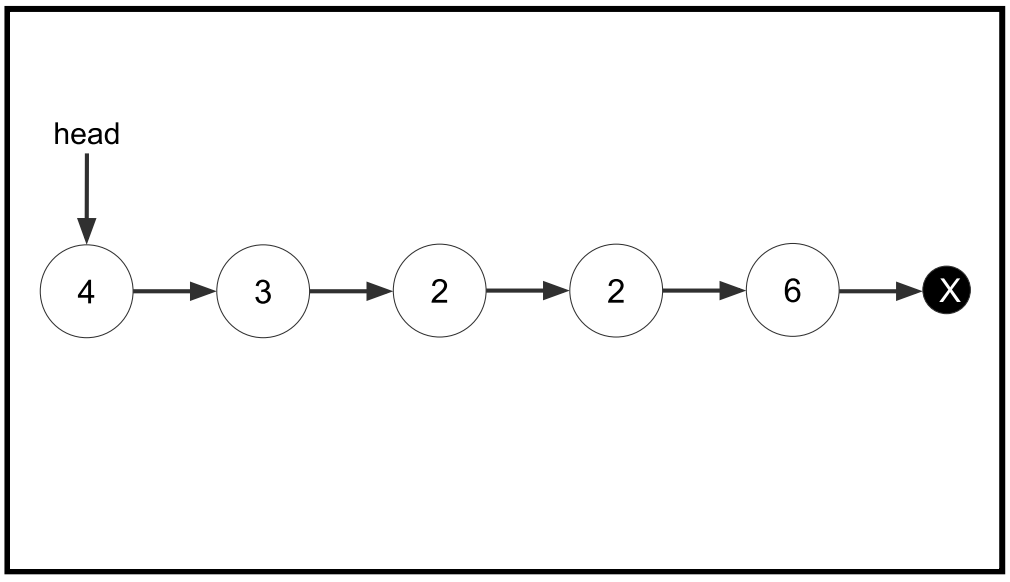
This question can have more than one answer and a correct answer will identify all the issues in the code.

For example, if you think that A and C are the correct answers, your answer will be: AC

If you think that D alone is the correct answer, your answer will be: D

Question 11 (1 mark)

A program contains a linked list that looks like this:



Assuming the following code for an node struct:

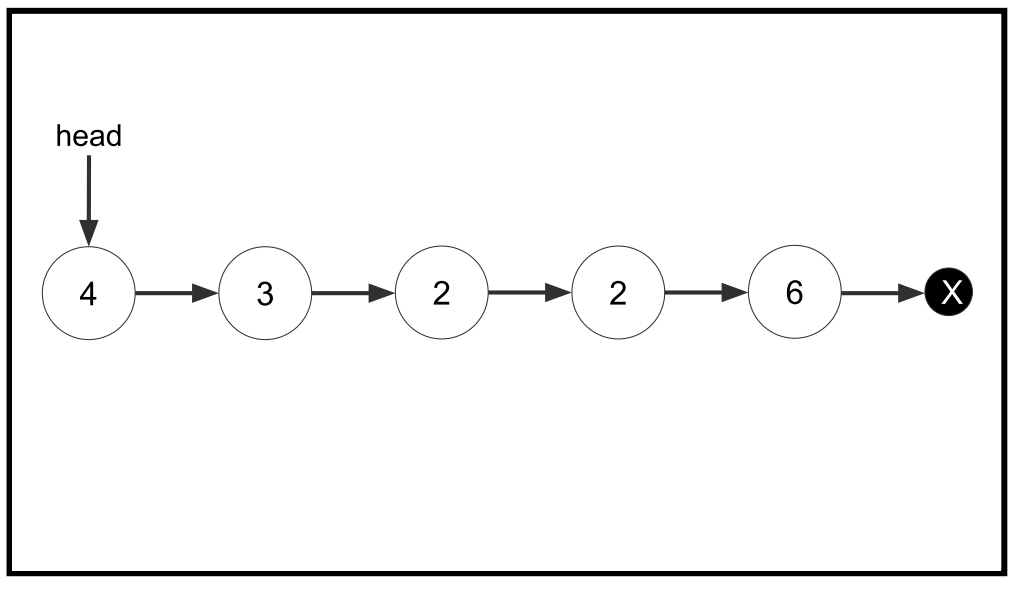
```
struct node {
    struct node *next;
    int data;
};
```

The following code is then executed:

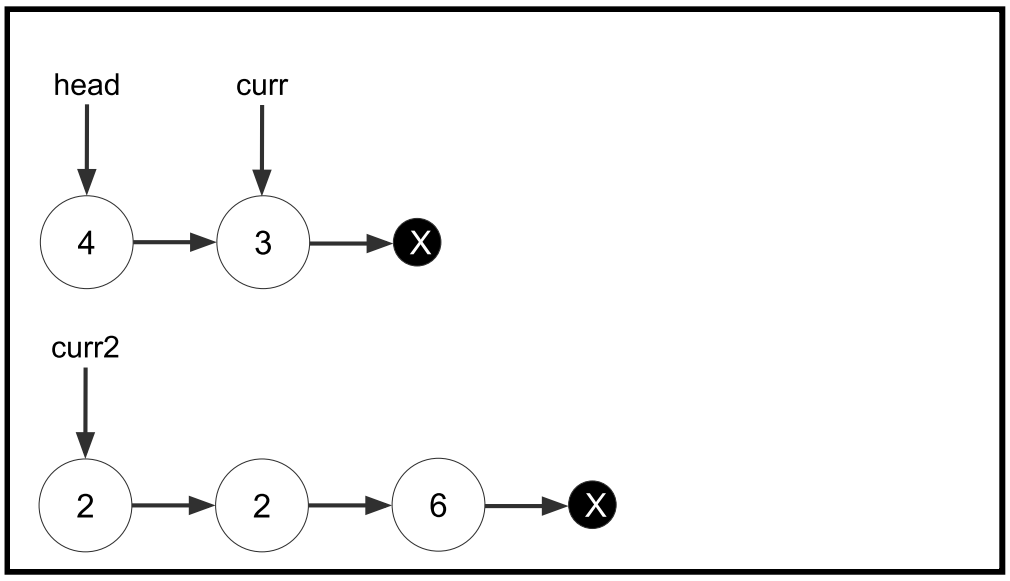
```
struct node *curr = head;
while (curr->data != curr->next->data) {
        curr = curr->next;
}

struct node *curr2 = curr->next;
curr->next = NULL;
```

Which of the following diagrams best represnts the new state of the linked list?

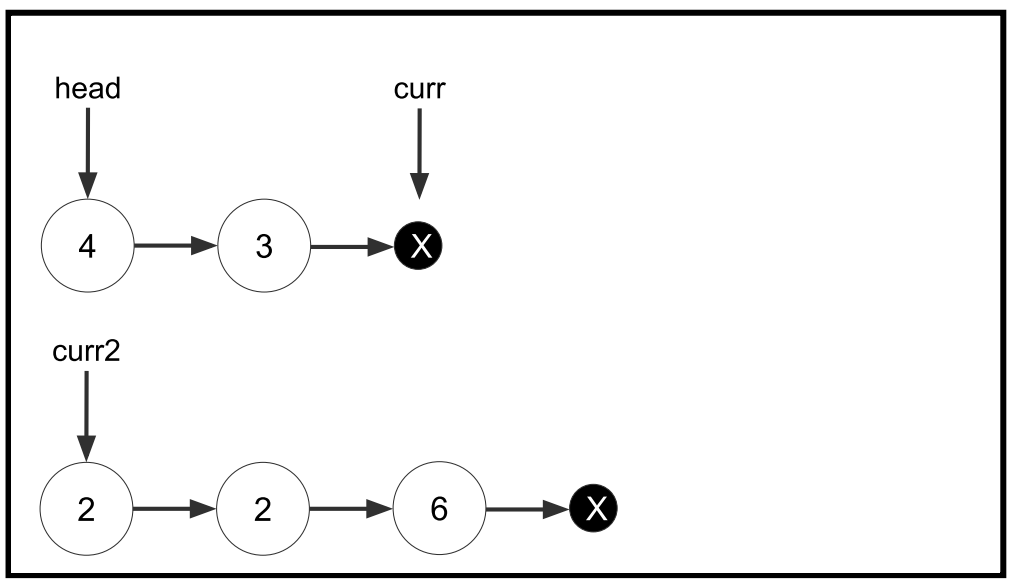
- **A**



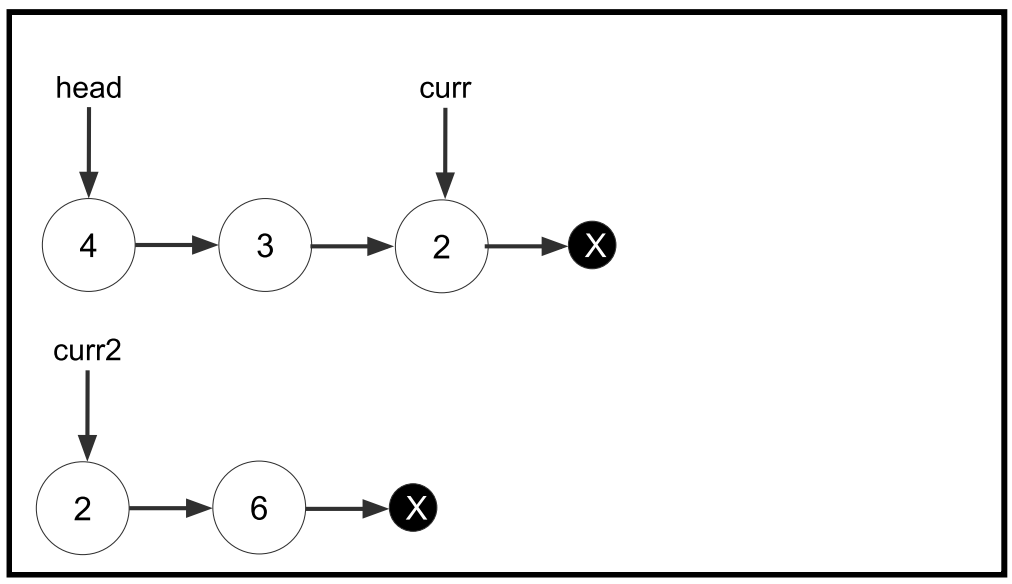
- **B**



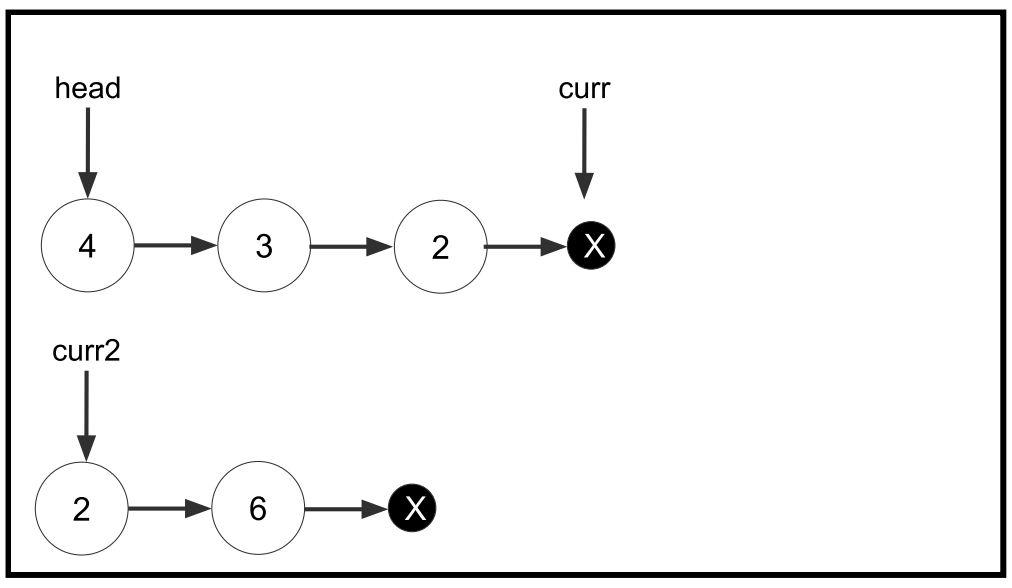
- **C**



- **D**



- **E**



Enter this answer in the file `exam_mc.txt`, after the `[q11]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Your answer should be either A, B, C, D, or E.

Do not enter any extra characters. Do not enter any explanation.

Question 12 (1 mark)

Imagine a file named `part1_q12.c` (which has not been provided) contains this C Code:

```c
#include <stdio.h>

void work(int num, int nums[10], int *pointer);

int main(void) {
    int my_number;
    int my_numbers[10];
    int *num_pointer;
    my_number = 5;
    my_numbers[4] = 3;
    num_pointer = &my_number;
    work(my_number, my_numbers, num_pointer);
    printf("DONE\n");
}

void work(int num, int nums[10], int *pointer) {
    num = 3;
    nums[4] = 12;
    *pointer = 5;
    pointer = nums;
    *pointer = 4;
}
```

Imagine `part1_q12.c` is compiled with dcc on a CSE machine like this:

```
$ dcc part1_q12.c -o part1_q7
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./part1_q12
```

When the code reaches line 13 (but has not executed it yet), what is the value of the variable my_number?

Your answer should be the value of an integer, for example: 99

Enter this answer in the file `exam_mc.txt`, after the `[q12]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Do not enter any extra characters. Do not write \n for a newline character. Do not enter any explanation.

Question 13 (1 mark)

Imagine a file named `part1_q13.c` (not provided) contains this C Code:

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

int main(int argc, char *argv[]) {
    printf("%s", argv[2]);
    printf("%d", argc);
}
```

Imagine `part1_q13.c` is compiled with dcc on a CSE machine like this:

```
$ dcc part1_q13.c -o part1_q13
```

The code is then run like so:

```
$ ./part1_q13 [SECRET]
has
4
```

It compiles successfully. No errors or warnings are produced by dcc.

Give an example of an input that could replace **[SECRET]** that would produce the shown output.

Enter this answer in the file `exam_mc.txt`, after the `[q13]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Question 14 (1 mark)

Imagine a file named `part1_q14.c` (not provided) contains this C Code:

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

struct node {
    int data;
    struct node *next;
};

struct node *createNode(int newData, struct node *newNext);

int main(void) {
    struct node *firstnode = createNode(1, NULL);
    struct node *head = firstnode;
    struct node othernode;
    othernode.data = 2;
    othernode.next = head;
    head = &othernode;
    free(firstnode);
}

struct node *createNode(int newData, struct node *newNext) {
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = newData;
    newNode->next = newNext;
    return newNode;
}
```

Imagine `part1_q14.c` is compiled with dcc on a CSE machine like this:

```
$ dcc part1_q14.c -o part1_q14
```

It compiles successfully. No errors or warnings are produced by dcc.

Which of the following statements are true about this code:

- A: There is exactly one memory leak in this program. firstnode hasn't been freed
- B: There is exactly one memory leak in this program. othernode hasn't been freed.
- C: There are two memory leaks in this program. Both firstnode and othernode haven't been freed.
- D: There is no memory leak in this program

Enter this answer in the file `exam_mc.txt`, after the `[q14]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

This question can have only one answer.

For example, if you think that A is the correct answer, your answer will be: A

Question 15 (1 mark)

We would like to build a struct to store some information about our favourite movies. We know we would like to store the name of the movie, it's playing length in minutes, and a pointer to it's sequel (if there is one).

Which of the following struct definitions would be best to use for this purpose?

- A:

```c
struct movie {
    char name[MAX_NAME_LENGTH];
    int length;
    struct movie sequel;
};
```

- B:

```c
struct movie {
    int name[MAX_NAME_LENGTH];
    int year;
    struct movie *sequel;
};
```

- C:

```
struct movie {
    string name;
    double length;
    struct movie *sequel;
};
```

- D:

```
struct movie {
    char name[MAX_NAME_LENGTH];
    int *length;
    struct movie sequel;
};
```

- E:

```
struct movie {
    char name[MAX_NAME_LENGTH];
    *struct movie sequel;
    int length;
};
```

Enter the answer(s) in the file `exam_mc.txt`, after the `[q15]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

This question can have only one answer and a correct answer will be the most appropriate struct for this purpose.

For example, if you think that A is the correct answer, your answer will be: A

Question 16 (1 mark)

Imagine a file named `part1_q16.c` (not provided) contains this C Code:

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

struct node {
    int data;
    struct node *next;
};

struct node *join_lists(struct node *before, struct node *after) {
    if (after == NULL) {
        return NULL;
    }
    struct node *curr = before;
    while (curr->next != NULL) {
        curr = curr->next;
    }
    curr->next = after;

    return before;
}
```

This code joins two linked lists into one list. It contains bugs. Which of the following cases will not produce a correct result?

- A: When `before` is a linked list with 3 elements, and `after` is an empty linked list .
- B: When `before` is an empty linked list, and `after` is a linked list with 2 elements.
- C: When `before` is an empty linked list, and `after` is an empty linked list.
- D: When `before` is an empty linked list, and `after` is a linked list with 1 element.
- E: When `before` is a linked list with 1 elements, and `after` is a linked list with 2 elements.

Enter this answer in the file `exam_mc.txt`, as a series of letters, after the `[q16]`, inside the curly brackets.

For example, if you belive that the first and second cases would not work, you should answer **AB**.

For example, if you belive that only the last case would not work, you should answer **E**.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Question 17 (1 mark)

Which one of the following four functions will return a value for all valid inputs it could be given? In other words, your answer will be the function that **will not** raise an error or recurse infinitely.

- A:

```
// assume x is any integer.
int recursive_a(int x) {
    if (x == 1) return 1;
    return recursive_a(x-1) + recursive_a(x-1);
}
```

- B:

```
// assume n is a valid linked list
int recursive_b(struct node *n) {
    if (n == NULL) return 0;
    return recursive_b(n->next) + 1;
}
```

- C:

```
// assume x is any integer.
int recursive_c(int x) {
    if (x > 0) return recursive_c(x - 1);
    if (x < 0) return recursive_c(x + 1);
}
```

- D:

```
// assume array is a valid array of integers,
// assume i is any integer.
void recursive_d(int array[128], int i) {
    if (i < 0) {
        return;
    }
    array[i] = i;
    recursive_d(array, i - 1);
}
```

Enter the answer(s) in the file `exam_mc.txt`, after the `[q17]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

For example, if you think that A is the function that will return a value for any given input, answer **A**. Only one of the functions will return a correct value for all possible inputs it could be given.

Question 18 (1 mark)

Imagine you are working on a multi-file project, involving three files:

- `main.c`
- `stack.h`
- `stack.c`

The stack files should implement a typedef called `typedef struct stack *Stack` in `stack.h`, and implement the fields of the `struct stack` in `stack.c` The main file will use stack.c and stack.h. The code follows all the conventions we have taught you in COMP1511 regarding naming and ADT writing.

Which of the following statements are correct?

- A: `main.c` must include the line `#include <stack.h>`
- B: `stack.c` must include the line `#include "stack.h"`
- C: `main.c` could include the lines `struct stack s = malloc(sizeof(struct stack)); s->next = NULL;`
- D: The following command would compile this code: `dcc -o program stack.c main.c`
- E: A function in `stack.c`, with no prototype in `stack.h` could be called inside of `main.c`

Enter this answer in the file `exam_mc.txt`, as a series of letters, after the `[q18]`, inside the curly brackets.

For example, if you belive that the first and second cases would not work, you should answer **AB**.

For example, if you belive that only the last case would not work, you should answer **E**.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Question 19 (1 mark)

A program contains a stack with the following functions visible in its header file:

```
// Adds a new item to the stack
Stack push(Stack q, int addItem);

// Removes and returns the item that should leave the stack next
int pop(Stack q);
```

The following code is run as part of a program:

```
push(myStack, 1);
push(myStack, 2);
int a = pop(myStack);
push(myStack, 3);
push(myStack, 4);
int b = pop(myStack);
push(myStack, 5);
```

What are the values of a and b?

Enter this answer in the file `exam_mc.txt`, after the `[q19]`, inside the curly brackets.

You can check the format of your answer is correct with `1511 exam-autotest exam_mc`.

Your answer should be two integers separated by a space.

For example, if you think that the values are 100 for a and 200 for b (in that order), your answer will be: 100 200

Question 20 (1 mark)

Imagine a file named `part1_q20.c` (which has not been provided) contains this C Code (amongst other code):

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[]) {
4      int i = 0;
5      while (argv[1][i+1] != '\0') {
6          if ([SECRET]) {
7              printf("argv[1] contains a string with a lowercase next to an uppercase!");
8              break;
9          }
10         i++;
11     }
12 }
```

The answer to this question is a piece of code that could replace **[SECRET]** on line 6 of this code. This piece of code should test if `argv[1]` contains a lowercase letter directly before an uppercase letter.

For instance, you should ensure that the argument "aBcd" prints something out. The argument "ABCD" or "abcd" should not print anything out.

Your answer should ensure that dcc compiles the program correctly; and that there will be no runtime errors.

You may not change any other lines of the program.

You can check your answers are in the correct format with `1511 autotest-exam exam_mc`
You can submit the question with `give cs1511 exam_mc exam_mc.txt`

Practical Questions

Question 1

Achieving >= 50% marks in this question, or question 3, is sufficient to pass the Arrays Hurdle.

(question codename: array_count_neutral)

`count_neutral_rows` will be passed a two dimensional array with 4 rows and `length` columns. Add code so that **count_neutral_rows** returns the number of neutral rows in the array. A neutral row is a row that adds up to 0.

For example if the array contains these elements:

```
{16, 16, 16, 16, 16},
{2,   2,  2, -4, -2},
{2,  -2,  1, -4,  3},
{2,   2,  1, -3, -2},
```

Your function should return **3**, since every row except the first adds up to 0.

For example if the array contains these elements:

```
{17},
{2},
{0},
{4},
```

Your function should return **1**, because only the third row adds up to 0.

Testing

exam_q1.c also contains a simple **main** function which allows you to test your **count_neutral_rows** function.

Your **count_neutral_rows** function will be called directly in marking. The main function is only to let you test your **count_neutral_rows** function

Assumptions/Restrictions/Clarifications.

Your function should return a single integer; between 0 and 4 (inclusive).

`count_neutral_rows` should not change the array it is given.

`count_neutral_rows` should not call scanf (or getchar or fgets).

`count_neutral_rows` can assume the array always has 4 rows, and has at least 1 column.

`count_neutral_rows` should not print anything. It should not call printf.

Your starter code contains a main function. You can change it for your own testing, but it will not be autotested or marked.

You can autotest this code with `1511 autotest-exam exam_q1`
You can submit this code with `give cs1511 exam_q1 exam_q1.c`
You can check your submission has been accepted with `1511 classrun -check exam_q1`
You can see your previous autotests here

Question 2

Achieving >= 50% marks in this question, or question 4, is sufficient to pass the Linked Lists Hurdle.

(question codename: list_count_changes)

Note **exam_q2.c** uses the following familiar data type:

```
struct node {
    struct node *next;
    int          data;
};
```

count_changes is given one argument: **head** a pointer to the first node of a linked list.

count_changes should return the number of times an element in the list is different to the element before it.

For example if the linked list contains these elements:

```
16, 16, 17, 16
```

count_changes should return **2**, because the list goes from 16 to 17, and then goes from 17 to 16 again.

For example if the linked list contains these elements:

```
12, 12, 8, 6, 8
```

count_changes should return **3**, because every pair of numbers except the first contains different elements.

Testing

exam_q2.c also contains a **main** function which allows you to test your **count_changes** function.

This main function:

- converts the command-line arguments to a linked list
- assigns a pointer to the first node in the linked list to **head**

- calls **count_changes(head)**
- prints the result.

Do not change this main function. If you want to change it, you have misread the question.

Your **count_changes** function will be called directly in marking. The main function is only to let you test your **count_changes** function

Here is how the main function allows you to test **count_changes**:

```
$ dcc exam_q2.c -o exam_q2
$ ./exam_q2 3 1 4
2
$ ./exam_q2 3 1
1
$ ./exam_q2 2 4 6 42
3
$ ./exam_q2 1 1 1 1
0
$ ./exam_q2 42
0
$ ./exam_q2
0
```

Assumptions/Restrictions/Clarifications.

count_changes should return only a single integer and this integer must be greater than or equal to 0.

count_changes should not change the linked list it is given.

count_changes should not change the next or data fields of list nodes.

count_changes should not use arrays.

count_changes should not call malloc.

count_changes should not call scanf (or getchar or fgets).

count_changes should not print anything. It should not call printf.

Do not change the definition of **struct node**.

Do not change the supplied **main** function. It will not be tested or marked.

You can autotest this code with `1511 autotest-exam exam_q2`
You can submit this code with `give cs1511 exam_q2 exam_q2.c`
You can check your submission has been accepted with `1511 classrun -check exam_q2`
You can see your previous autotests here

Question 3

Achieving >= 50% marks in this question, or question 1, is sufficient to pass the Arrays Hurdle.

(question codename: coprimes)

Write a C program **exam_q3.c** which reads integers from standard input until it reaches end-of-input. The integers will be unique, in increasing order, and larger than 0.

Your program should print out that list again, but replace any number that is a multiple of another number in the list with a '-1'.

Make your program produce exactly the same output as the examples below.

```
$ dcc exam_q3.c -o ./exam_q3
$ ./exam_q3
2
3
6
7
9

Ctrl-D
2
3
-1
7
-1
$ ./exam_q3
12
18
24
30
36

Ctrl-D
12
18
-1
30
-1

$ ./exam_q3
1
2
4
6
8
10

Ctrl-D
1
-1
-1
-1
-1
-1
```

Assumptions/Restrictions/Clarifications.

Your program must read until the end of-input. End of input is signalled on a Linux terminal by typing the **Ctrl** and **d** keys together. This is what `Ctrl-D` indicates in the above examples.

You can assume your input contains at least one integer.

You can assume your input contains no more than 10000 integers.

You can assume each integer will be strictly larger than the integer before it, and that no integer will be less than 1.

No error checking is necessary.

You are free to write this program in any way you wish: there is no specific function that you need to implement. Note that your program will need to have a `main` function.

You can autotest this code with `1511 autotest-exam exam_q3`
You can submit this code with `give cs1511 exam_q3 exam_q3.c`
You can check your submission has been accepted with `1511 classrun -check exam_q3`
You can see your previous autotests here

Question 4

Achieving >= 50% marks in this question, or question 2, is sufficient to pass the Linked Lists Hurdle.

(question codename: list_insert_middle)

insert_middle is given two arguments, **value** and **head**. **head** is the pointer to the first node in a linked list.

Add code to **insert_middle** so that it creates a new list node (using `malloc`) containing **value** and places it in the middle of the given list.

If the list has an even number of nodes, it should be inserted such that there are an equal number of nodes before and after it. If the list has an odd number of nodes, it should be inserted such that there is one more node before it than there is after it.

insert_middle should return a pointer to the new list.

For example if **value** is **12** and the linked list contains these 3 elements:

```
16, 7, 8
```

insert_middle should return a pointer to a list with these elements:

```
16, 7, 12, 8
```

Testing

exam_q4.c also contains a **main** function which allows you to test your **insert_middle** function.

This main function:

- converts the command-line arguments to a linked list
- assigns a pointer to the first node in the linked list to **head**
- reads an integer from standard input and assigns it to **n**
- reads a second integer from standard input and assigns it to **value**
- calls **insert_middle(n, value, head)**
- prints the result.

Do not change this main function. If you want to change it, you have misread the question.

Your **insert_middle** function will be called directly in marking. The main function is only to let you test your **insert_middle** function

```
$ dcc exam_q4.c -o exam_q4
$ ./exam_q4 16 7 8
12
[16, 7, 12, 8]
$ ./exam_q4 1 16 7 8
12
[1, 16, 12, 7, 8]
$ ./exam_q4 42
16
[42, 16]
$ ./exam_q4
2
[2]
```

Assumptions/Restrictions/Clarifications.

insert_middle should not use arrays.

insert_middle should not call scanf (or getchar or fgets).

insert_middle should not print anything. It should not call printf.

Do not change the supplied **main** function. It will not be tested or marked.

You can autotest this code with `1511 autotest-exam exam_q4`
You can submit this code with `give cs1511 exam_q4 exam_q4.c`
You can check your submission has been accepted with `1511 classrun -check exam_q4`
You can see your previous autotests here

Question 5

This question will not satisfy either hurdle

(question codename: similar_words)

Write a C program `exam_q5.c` which reads in two lines of input, and a command line argument.

Your program should count the number of differences (ignoring case) between the program argument, and each line of input. It should then print the smaller number of differences, as well as the string that had that number of differences.

You can assume that both lines, and the program's argument are of the same length.

If two lines have the same number of differences, you should print the first line.

For example, if the argument was "and", and two lines were "nod" and "ANT", it would look like this:

Index	0 1 2
Argument:	a n d
Line 1:	n o d
Line 2:	A N T

You would print out "1 ANT", since it's first and second letters match (ignoring case) with the argument. While "nod" is the same case as "and", it only has one letter match, which is less than "ANT" (which has two letters matching); so it is not printed out.

For example:

```
$  dcc exam_q5.c -o exam_q5
$ ./exam_q5 hello
happy
hallo
1 hallo
$ ./exam_q5 testing
lasting
addINGs
2 lasting
$ ./exam_q5 TEsting
Nesting
TEsting
0 TEsting
$ ./exam_q5 testing
TeStInG
testing
0 TeStInG
```

Assumptions/Restrictions/Clarifications.

You can assume the argument contains no more than 256 characters.

You can assume neither input line contains more than 256 characters.

You can assume your input always contains two lines, ending in a newline ('\n')

No error checking is necessary.

You are free to write this program in any way you wish: there is no specific function that you need to implement. Note that your program will need to have a `main` function.

You can autotest this code with `1511 autotest-exam exam_q5`
You can submit this code with `give cs1511 exam_q5 exam_q5.c`
You can check your submission has been accepted with `1511 classrun -check exam_q5`
You can see your previous autotests here

Question 6

This question will not satisfy either hurdle

(question codename: exam_fix_html)

This challenge does not provide comprehensive test cases

You have been provided some autotests to check your output is coherent, and that it is in a correct format. Passing the autotests does not guarantee that you have solved this challenge -- your own testing will be required.

HTML is the language which websites on the internet are written in. HTML uses opening and closing tags to define the structure and formatting of a website. For example, `<p>` is an opening paragraph tag and `</p>` is a closing paragraph tag. The slash in the second tag shows that it is a closing tag. Tags must be opened and closed in order, so if an `<i>` tag is opened inside a `<b>` tag, the `</i>` closing tag must come before the `</b>` closing tag.

Sometimes, when we write HTML, we forget to add closing tags. In this exercise we will make a function which can add missing closing tags in our HTML for us.

In this exercise you will be modifying a linked list of HTML tags. Each node of the linked list contains an array of characters. The tag is a string in this array.

```
struct tag {
    char tag[MAX_TAG_NAME];
    struct tag *next;
};
```

The tag string will always include an opening anglebracket in the first character. If the tag is a closing tag it will have a slash after the opening anglebracket. The tag will always have only a single lowercase letter then a closing anglebracket after that.

```
<p> -> <i> -> </p> -> NULL
```

The above list represents some HTML which was poorly written. Inside the `<p>` tag, an `<i>` tag was opened, but never closed. In this case, the `fix_html` function should add the necessary closing tags in order to fix this HTML.

```
<p> -> <i> -> </i> -> </p> -> NULL
```

The `fix_html` function should not delete any of the tags in the linked list of tags it is given. It should insert new closing tags in order to make the HTML correct. HTML is considered correct if every opening tag has a closing tag and tags are properly nested. Each tag should have a closing tag before its parent's closing tag. Each closing tag in the sequence must be the most recently opened, but not already closed tag.

Here are some examples of HTML lacking closing tags being fixed. This is what the main function should print when the `fix_html` function is completed.

```
$ ./fix_html
Before fix: <p><b></p>
After fix: <p><b></b></p>

Before fix: <p><i><b><i></p>
After fix: <p><i><b><i></i></b></i></p>

Before fix: <b><i></i><p></b>
After fix: <b><i></i><p></p></b>
```

Assumptions

- You can assume you will never be given an empty linked list.
- You can assume that you will never be given a list of tags which has a closing tag before any matching open tag. Only closing tags will be missing, not opening tags.
- You can assume that there will only be one lowercase letter inside the tag string.
- You can assume you will never be given a linked list with more than 100 nodes representing broken HTML.
- Your function should call `malloc` to create new tag nodes and insert them in the linked list.
- The main function will not be marked. Only your `fix_html` function and any other functions it calls will be marked.

You can autotest this code with `1511 autotest-exam exam_q6`
You can submit this code with `give cs1511 exam_q6 exam_q6.c`
You can check your submission has been accepted with `1511 classrun -check exam_q6`
You can see your previous autotests here

Question 7

This question will not satisfy either hurdle

(question codename: exam_word_wrap)

This challenge does not provide comprehensive test cases

You have been provided some autotests to check your output is coherent, and that it is in a correct format. Passing the autotests does not guarantee that you have solved this challenge -- your own testing will be required.

Write a C program (inside the file `exam_q7.c`) which reads in a paragraph of text, and arranges the words into lines such that no line is longer than the specified line width.

This program is given two command line arguments: the maximum width for each line, and a single character representing the output style to use (which will either be 'L' or 'R').

This program is then given a paragraph of text, in the form of a series of characters on standard input, across multiple lines. This program should read characters from standard input, until it reaches the end of the input (EOF).

This program should then print out the words it has read across multiple lines, such that each line has as many words as possible without going over the maximum line width.

A *word* is defined to be a consecutive sequence of letters, digits and/or punctuation surrounded by one or more spaces on either side (although the first word of an input line need not be preceded by a space, and the last word of an input line need not be followed by a space). Thus the first word of the question that you are reading now is "Write", and the fifth word is **(inside** (including the bracket).

You must arrange the words into lines so that each line except for the last contains as many words as possible. Words within each line must be separated by at least one space (see below for details), and the total length of each line (including spaces) must not exceed the maximum line width.

Of course some lines may have total length less than the maximum line width, and so the question remains how to display such lines on the screen. You will be asked to use one of two styles: left flushed or right flushed.

These styles of output are explained below, and are illustrated in the examples below.

Left-flushed output (the 'L' option)
There must be precisely one space between any two adjacent words on a line, and the first character of the first word on each line must be placed as far left as possible (i.e. there are no spaces at the start of each line).

Right-flushed output (the 'R' option)
There must be precisely one space between any two adjacent words on a line, and the last character of the last word on each line must be placed as far right as possible (i.e. in the *w*th column of output, where *w* is the maximum line length).

Examples

Left-aligned text (width 30):

```
$ dcc -o exam_exam_q7 exam_exam_q7.c
$ ./exam_exam_q7 30 L
Muskrats spend much of their time in the water and are well suited for their
semi-aquatic life, both in and out of water. Muskrats can swim underwater for
12 to 17 minutes.
Ctrl-D
Muskrats spend much of their
time in the water and are well
suited for their semi-aquatic
life, both in and out of
water. Muskrats can swim
underwater for 12 to 17
minutes.
```

Right-aligned text (width 30):

```
$ dcc -o exam_q7 exam_q7.c
$ ./exam_q7 30 R
Muskrats spend much of their time in the water and are well suited for their
semi-aquatic life, both in and out of water. Muskrats can swim underwater for
12 to 17 minutes.
Ctrl-D
  Muskrats spend much of their
time in the water and are well
 suited for their semi-aquatic
      life, both in and out of
      water. Muskrats can swim
       underwater for 12 to 17
                     minutes.
```

Note that the words in the input may be surrounded by multiple spaces, e.g.

Left-aligned text (width 20):

```
$ dcc -o exam_q7 exam_q7.c
$ ./exam_q7 20 L
The novelist Matteo Bandello   observed   Leonardo at work      and
   wrote that   some days
 he would paint from     dawn    till dusk    without  stopping
     to eat, and then not paint for three or
 four days at a time.
Ctrl-D
The novelist
Matteo Bandello
observed Leonardo
at work and wrote
that some days he
would paint from
dawn till dusk
without stopping
to eat, and then
not paint for
three or four days
at a time.
```

Right-aligned text (width 20):

```
$ dcc -o exam_q7 exam_q7.c
$ ./exam_q7 20 R
The novelist Matteo Bandello   observed   Leonardo at work      and
   wrote that   some days
 he would paint from     dawn    till dusk    without  stopping
     to eat, and then not paint for three or
 four days at a time.
Ctrl-D
      The novelist
   Matteo Bandello
 observed Leonardo
 at work and wrote
 that some days he
  would paint from
    dawn till dusk
  without stopping
  to eat, and then
     not paint for
three or four days
        at a time.
```

Assumptions/Restrictions/Clarifications.

- You can assume that each line of input will be no more than 500 characters long.
- You can assume that each line of input has at least one word.
- You can assume that the maximum output line width will be no longer than 100 characters.
- You can assume that no word will be longer than the maximum line width.
- You can assume there will be no more than 10000 words in total across all of the lines.
- You can assume that each line of output will have at least two words, i.e. that you will never have a line of output with just one word on it.

You can autotest this code with `1511 autotest-exam exam_q7`
You can submit this code with `give cs1511 exam_q7 exam_q7.c`
You can check your submission has been accepted with `1511 classrun -check exam_q7`
You can see your previous autotests here

Question 8

This question will not satisfy either hurdle

(question codename: total_ordering)

This challenge does not provide comprehensive test cases

You have been provided some autotests to check your output is coherent, and that it is in a correct format. Passing the autotests does not guarantee that you have solved this challenge -- your own testing will be required.

This task is all about bringing order to chaos. You will be given multiple lines on standard input consisting of a number, then two words (a word being a string not including spaces). Each line will be of the form "N word1 word2", which represents a statement of the form "word1 is N places before word2". Your task is to take all of these statements, and print out the correct ordering they describe.

A simple example would be:

```
1 first second
1 second third
```

This an input which describes the ordering "first second third". Because "first" is 1 place before "second", and "second" is 1 place before "third".

Note that the ordering may include redundant (i.e. useless) information (for instance, the same line could be repeated; or two lines could contain identical information). The following example demonstrates a valid input with redundant information:

```
1 first second
1 second third
1 first second
2 first third
```

The last two lines are redundant because "1 first second" is repeated, and "2 first third" describes the same thing as the first two lines together imply.

There may also be gaps between two inputs. In this case, you should ignore the gaps.

```
2 one three
3 three six
```

Should print out, exactly:

```
one three six
```

You are guaranteed, however, that the statements will always correctly describe a single, unique ordering. You will never be told that two different words could occupy the same space, and the same word will never be in two different places. You will also never end up with statements that contradict eachother. For instance, the following inputs would be invalid:

```
1 first second
1 first also_second
```

The above is invalid because "second" and "also_second" share a space.

```
1 first second
2 first third
1 also_second third
```

The above is invalid because "second" and "also_second" share a space; though it's not as obvious.

```
1 first second
1 second first
```

The above is invalid because it's contradictory -- first cannot both be before and after second.

A more complex ordering might be:

```
2 COMP3231 COMP9242
1 COMP1511 COMP1521
1 COMP3231 COMP3891
3 COMP1521 COMP9242
2 COMP1521 COMP3891
3 COMP1511 COMP3891
```

Which describes the ordering "COMP1511 COMP1521 COMP3231 COMP3891 COMP9242".

Your program should behave exactly like this:

```
dcc -o q8 q8.c
$ ./q8
1 Philosophers_Stone Chamber_Of_Secrets
4 Chamber_Of_Secrets Halfblood_Prince
2 Goblet_Of_Fire Halfblood_Prince
2 Order_Of_The_Phoenix Deathly_Hallows
3 Chamber_Of_Secrets Order_Of_The_Phoenix
2 Prisoner_Of_Azkaban Order_Of_The_Phoenix
1 Halfblood_Prince Deathly_Hallows
4 Fantastic_Beasts Philosophers_Stone
Ctrl-D
Fantastic_Beasts Philosophers_Stone Chamber_Of_Secrets Prisoner_Of_Azkaban Goblet_Of_Fire Order_Of_The_Phoenix
Halfblood_Prince Deathly_Hallows
```

Assumptions/Restrictions/Clarifications.

You will never receive any integers smaller than 1.

You will never receive a word that is longer than 50 characters.

There is no limit on the number of facts you could be given, nor on the number of items you could be given to order.

You can autotest this code with `1511 autotest-exam exam_q8`
You can submit this code with `give cs1511 exam_q8 exam_q8.c`
You can check your submission has been accepted with `1511 classrun -check exam_q8`
You can see your previous autotests here

Pets

This exam contains some motivational pet memes. The pets involved all live with current tutors of the course. To prevent distraction, you can choose whether to enable these pets. Check this box to enable pets:

Enable Pets? ☐