Question 1. [9 MARKS]

In the box beside each set of `bash` statements, show the output when the statements are executed. If running these statements would result in an error, print ERROR and give a brief explanation.

Part (a) [2 MARKS]

```
first=Santa
last=Clause
name=first+last
echo name is $name
echo 'last is $last'
```

name is first+last
last is $last

Part (b) [1 MARK]

```
# The current working directory
# contains the files a1 a2 b c

if [ "a1 a2" = "a*" ]
then
    echo match
else
    echo "a*"
fi
```

a*

Part (c) [1 MARK]

```
# The current working directory
# contains the files a1 a2 b c

files="a*"
if [ "a1 a2" = $files ]
then
    echo match
else
    echo $files
fi
```

Error: The problem is that $files expands to more than one word and the if fails.

Part (d) [2 MARKS]

In the current directory, some of the files contain the string `FIX ME`. Write a shell command that counts the number of lines containing this phrase in the files ending in `.c`

```
grep "FIX ME" *.c | wc
```

Part (e) [3 MARKS]

Write a shell loop to create 3 files named `test_one`, `test_two`, `test_three` that each contain the word `one`, `two`, or `three` respectively. (Hardcoding the answer with no loop will receive 0 marks.)

```
for suffix in one two three
do
  echo $suffix > test_$suffixs
done
```

Question 2. [5 MARKS]

Part (a) [2 MARKS]

In the box beside the code fragment below, provide the output. If the code results in an error before anything is printed, write ERROR and give a brief explanation.

```
char s[6] = "01234";
char * t = "XX";
strncpy(s+2, t, 3);
printf("%s\n", s);
```

01XX

Part (b) [1 MARK]

What is the type of `&t`?

Part (c) [2 MARKS]

Provide the output of the following program.

```
int double1(int x) {
    x = x + x;
    return x;
}

int double2(int *x) {
    *x = (*x) + (*x);
    return *x;
}


int main() {
    int i = 10;
    int j = 0;

    j = double1(i);
    printf("i = %d, j = %d\n", i, j);       //Output:  i = 10, j = 20

    i = 10;
    j = double2(&i);
    printf("i = %d, j = %d\n", i, j);       //Output:  i = 20, j = 20
    return 0;
}
```

Question 3. [4 MARKS]

Suppose the current working directory contains the following files, and that the `makefile` contains the rules in the box to the right:

```
a3          helpers.h  t2.txt
a3.c        makefile
helpers.c   t1.txt
```

```
1    all_tests: test1 test2

2    test1: a3
3           a3 t1.txt

4    test2: a3
5           a3 t2.txt

6    a3 : a3.o helpers.o helpers.h
7          gcc ${CFLAGS} -o a3 a3.o helpers.o

8    %.o : %.c helpers.h
9           gcc ${CFLAGS} -c $@
```

In the table below are four commands that are run sequentially (one after the other) from the shell. Complete the table by filling in the line numbers of the actions that are executed in and by filling in the names of files that are changed or created.

Command	Line numbers of executed actions	Names of Files Created or Changed
`make a3`	9, 9, 7	a3.o, helpers.o, a3
`make test2`	5	nothing changed (okay to say t2.txt changed)
`rm -rf a3`	No actions	a3 is removed
`make all_tests`	7, 3, 5	a3 (okay to say these changed:t1.txt, t2.txt)

Question 4. [12 MARKS]

Part (a) [7 MARKS] Consider the C program below which produces the output:

```
Even positions are 0246 and odd positions are 1357
Even positions are ACE and odd positions are BD
```

Complete the function `parity_strings` as described in the function comment. Your program should generate this output without making any changes to `main`. Your function must not change parameter `s`.

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

/* Return a pointer to an array of two strings. The first is the characters
   of string s that are in odd positions and the second is the characters from
   s that are in even positions */
char ** parity_strings(char * s) {
    // allocate space for the 2 pointers
    char ** result = malloc(2 * sizeof(char*));
    // allocate space for each of the two strings
    // their max length is int(strlen(s)/2.0 + .5) + 1
    // but it is okay if they do something simpler
    int is_odd, shorter_length, longer_length;
    is_odd = strlen(s) % 2;

    shorter_length = (int) (strlen(s) / 2);
    if (is_odd) {
        longer_length = shorter_length + 1;
    } else {
        longer_length = shorter_length;
    }

    result[0] = malloc(longer_length + 1);
    result[1] = malloc(shorter_length + 1);

    int i;
    for (i=0; i< shorter_length; i++) {
        result[0][i] = s[i*2];
        result[1][i] = s[i*2 + 1];
    }
    if (is_odd) {
        result[0][shorter_length] = s[shorter_length*2];
    }
    // now null terminate the strings
    result[0][longer_length] = '\0';
    result[1][shorter_length] = '\0';

    return result;
}
```

```
int main() {
    char s1[9] = "01234567";
    char ** r = parity_strings(s1);
    printf("Even positions are %s and odd positions are %s\n", r[0],r[1]);

    free_all(r);

    char * s2 = "ABCDE";
    r = parity_strings(s2);
    printf("Even positions are %s and odd positions are %s\n", r[0],r[1]);

    // this call is optional
    free_all(r);

    return 0;
}
```

Part (b) [1 MARK]

We explicitly stated that your function could not change parameter `s`. State when and why the program would fail if you didn't follow this instruction.

- Which call(s) to `paritystrings` would fail? (pick one)
  - ☐ Only the first call (on `s1`) would fail.
  - ☑ Only the second call (on `s2`) would fail.
  - ☐ Both calls would fail.

- Why would it fail? Because s2 is in read-only memory

Part (c) [1 MARK]

The program as written has a memory leak in the `main` function. Explain where this is? `r` is reassigned to the return value from the second `parity_strings` call and the first array returned by the first call is now lost.

Part (d) [3 MARKS]

Complete a function `free_all` that could be added to the program to fix the memory leak. Write the function here, but add the call(s) to the function into the main function on the previous page.

```
void free_all(char ** r) {
    free(r[0]);
    free(r[1]);
    free(r);
}
```

Print your name in this box.