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) [1 MARK]

```
v=mon
v="$v day"
echo $v
```

mon day

Part (b) [1 MARK]

```
y=thurs
y='$y day'
echo $y
```

$y day

Part (c) [1 MARK]

```
z="fri day"
for piece in "$z"
  do
    echo "$piece *"
  done
```

fri day *

Part (d) [1 MARK]

```
z="fri day"
for piece in $z
  do
    echo "$piece *"
  done
```

fri *
day *

Part (e) [2 MARKS]

Write a few lines of shell that would display the phrase `The current month is XXX` with the current month inserted instead of `XXX` Reminder: the default output from the command `date` looks as follows:

```
Wed Feb 20 16:36:00 EST 2013
```

```
set `date`
echo The current month is $2
```

Part (f) [3 MARKS]

Some of the files in the current working directory contain the phrase `FIX ME` in places that need correction. When we run the command `grep "FIX ME" *` the output contains one line for every line in the files containing `FIX ME`. The first few lines of output are shown here.

```
buxfer.c:FIX ME
commands.txt:FIX ME
commands.txt:line that needs changes // FIX ME
```

Write a single shell expression (using pipes) the counts the number of **different** files that need fixing (i.e. the number of files that contain this phrase.)

```
grep "FIX ME" * | cut -d ":" -f 1 | uniq | wc
```

Use this space for rough work or extra space for any answer. Clearly label anything you want marked.

Question 2. [5 MARKS]

Consider the following C code fragment.

```
char s[11] = "0123456789";
char * t = "source";
char * p = s + 3;
strncpy(s+2, t, 9);
printf("%s\n",s);
```

Part (a) [2 MARKS]

What is the output? If the code results in an error before anything is printed, write ERROR and give a brief explanation.

01source

Part (b) [1 MARK] What is the type of `&t` ? `char **`

Part (c) [1 MARK] What is the type of `*p` ? `char`

Part (d) [1 MARK] What is the **value** of `p[2]` after this fragment has executed? `'r'`

Question 3. [4 MARKS]

Suppose the current working directory contains the following files:

```
cheese  makefile
```

The contents of `makefile` is shown in the box to the right.

In the table below are three commands that are run sequentially (one after the other) from the shell. Complete the table by filling in the output that be printed to standard out and the names of files that are changed or created.

```
dinner: pizza salad

pizza: cheese topping.peppers
        echo making pizza
        cat cheese topping.peppers  > pizza

cheese:
        echo goey cheese > cheese

topping.%:
        echo yummy $@ > $@

salad:
        echo salad is healthy
```

Command	Output printed to standard output	Names of Files Created or Changed
make salad	salad is healthy	none
make pizza	making pizza	topping.peppers, pizza
make dinner	salad is healthy	none

Question 4. [12 MARKS]

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

```
first token is Craig and second token is Michelle
first token is Reid and second token is Karen
```

Complete function `split_on_comma` 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`. You may want to use function `char * strchr(const char *s, int c)` which returns a pointer to the first occurence of `c` in `s`. You do not need to check the return values of your system calls.

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

char ** split_on_comma(char * s) {

    /* This is one approach to solving this problem. It is not the
       only correct answer */

    int comma_index;
    comma_index = strchr(s, ',') - s;

    char ** result = malloc (2 * sizeof(char *));

    result[0] = malloc((comma_index + 1) * sizeof(char));
    //s contains comma_index digits and 1 comma and then the name
    // but we need space for the null-terminator
    result[1] = malloc(strlen(s) - comma_index);

    int i;
    for (i=0; i< comma_index; i++) {
        result[0][i] = s[i];
    }
    result[0][comma_index + 1] = '\0';

    // <= is so that we also copy the null terminator
    for (i=comma_index + 1; i <= strlen(s) ; i++) {
        result[1][i-comma_index - 1] = s[i];
    }
    return result;
}

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

```
int main() {
    char s1[15] = "Craig,Michelle";
    char ** r = split_on_comma(s1);
    printf("first token is %s and second token is %s\n",r[0],r[1]);

    free_all(r);

    char * s2 = "Reid,Karen";
    r = split_on_comma(s2);
    printf("first token is %s and second token is %s\n",r[0],r[1]);

    // may call free_all here again. Either way is ok.

    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 `split_on_comma` 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]

As written, the program has a memory leak in the `main` function. Explain where this is. `r` is reassigned to the return value from the second `split_on_commas` 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 `free_all` into the main function on the previous page.

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