Question 1. [7 MARKS]

In each subquestion below, fill in the box with the declaration for an appropriate mystery function so that the following code would compile without error. The subquestions are independent.

Part (a) [4 MARKS]

```
int mystery1(double *, int, char *);
```

```
double price;
int * position;
char ** products;
/* Assume that these variables are initialized properly here */

if (mystery1(&price, *position, products[3]) != 0) {
    fprintf(stderr, "Error with mystery 3\n");
}
```

Part (b) [3 MARKS]

```
char * mystery2(int, char ***);
```

```
int rating[3] = {4, 5, 10};
char ** items;
char name[30];
strncpy(name, mystery2(rating[2], &items), 29);
```

Question 2. [8 MARKS]

Part (a) [5 MARKS]

On the next page write a shell program `longer` that takes two filenames as command line arguments and prints to standard output the content of the file that is longer. By default, length is determined as the number of lines, but if the script is called with an optional -w argument (that must come before the filenames), it determines length based on the number of words instead.

For example, suppose that the two files are as shown here:

File1

```
File1 has only two lines
but lots of words
```

File2

```
File2
has
more
lines
```

If your function was called as `longer File1 File2`, it would print the contents of `File2`. If were called as `longer -w File1 File2`, it would print the contents of `File1`.

You may assume that the files corresponding to the arguments do exist and are readable.

```
#!/bin/sh

# prints to standard output the contents of whichever file is longer (has more lines)
# if the optional -w argument is present (before the filenames), the comparison
# is done based on the number of words in the file rather than the number of lines
```

```
if [ "$1" = "-w" ] ; then
    shift
    length1=‘wc -w "$1"‘
    length2=‘wc -w "$2"‘
else
    length1=‘wc -l "$1"‘
    length2=‘wc -l "$2"‘
fi

if [ $length1 -ge $length2 ]; then
  cat "$1"
else
  cat "$2"
fi
```

NOTE: The above solution doesn't quite work but we gave it full marks since we did not expect you to remember that wc also printed the filenames unless it was called with input from stdin. In order to do it correctly you want to say

```
length1=‘cat "$1" | wc -w‘
```

etc. We also gave full marks for a correct solution of course!

Part (b) [3 MARKS]

In order to test your script for part a, you would like to create five files named `file1, file2, file3, file4,` and `file5`. They should have the number of lines corresponding to their name. For example, `file4` should have four lines. Complete the shell script below that would create these files in a way that only the `for` loop would need changing to create more than five of these files. It doesn't matter exactly what you put on the lines of the test files as long as they aren't empty. You may assume that the files do not exist, or are empty before the script runs.

```
for i in 1 2 3 4 5; do
  #create file i

  count=0
  while [ $count -lt $i ]
  do
    count=‘expr $count + 1‘
    echo $i >> file$i
  done
done
```

Question 3. [4 MARKS]

Part (a) [1 MARK]

Two contstants `SIZE1` and `SIZE2` have valid, but unknown values. Suppose that `str2` is initialized to contain a string. Complete the line of code in the box below with a correct third argument to `strncpy` so that the result will be correct regardless of the values of `SIZE1`, `SIZE2` and `n`.

```
char str1[SIZE1];
char str2[SIZE2];

/* str2 is initialized to a valid string */
int n = strlen(str2);
```

```
strncpy(str1, str2, SIZE1);
```

Part (b) [2 MARKS]

Assuming the third argument to `strncpy` is correct, is it possible for `strlen(str1) > SIZE1`? If yes, give an example. If no, explain why not.

SOLUTIONS

Yes. If `SIZE1` = 5 and `str2` = `"this is too long"`, then once the copy happened with the above statement `str1` would contain the 5 characters `"this "` but no null-terminator. So the length would be calculated as the number of characters starting with the beginning of `str1` until a null-terminator was encountered in memory. Since `str2` is declared immediately after `str1`, there is a good chance that it might be the null terminator of `str2` but this isn't guaranteed.

Part (c) [1 MARK]

In the box below this code fragment, print the output of this code.

```
int a[5] = {1,2,3,4,5};
int * p = a;
p++;
printf("%d",p[2]);
```

```
4
```

Question 4. [6 MARKS]

Part (a) [3 MARKS]

Consider the C code below that defines a struct to represent a gift card. Write a function `adjust_amount` which takes a card and a charge and attempts to reduce the balance of the card by the charge. If there is enough money on the card, the balance is reduced and the function returns 0. If the card doesn't have enough money to cover the charge, then the balance is reduced to 0 and the amount that was successfully paid by the giftcard is returned.

```
#define MAXNAME 24
struct giftcard{
    char name[MAXNAME];
    double balance;
};

/* Attempt to reduce the balance by charge
 * If successful, return 0.
 * If the balance was insufficient, reduce the balance to zero and
 * return how much the gift card was charged. */
double adjust\_amount(struct giftcard * g, double charge) {
    if (g->balance >= charge) {
        g->balance -= charge;
        return 0;
    }
    double return_value = g->balance;
    g->balance = 0;
    return return_value;
}
```

Part (b) [3 MARKS]

Write C code that creates a struct to represent Karen Reid's gift card which has a balance of $3.98. Then call your function to spend $2.05 (enough for a starbuck's grande) on Karen's card.

```
struct giftcard g;
strncpy(g.name, "Karen Reid", MAXNAME);
g.balance= 3.98;

adjust_amount(&g, 2.05);
```