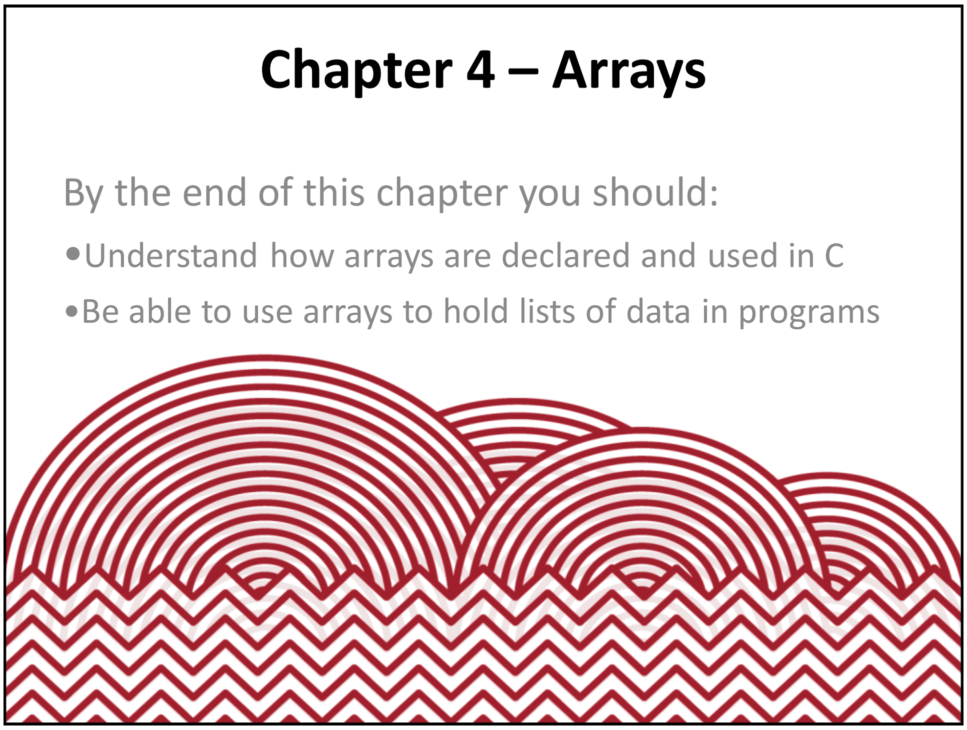
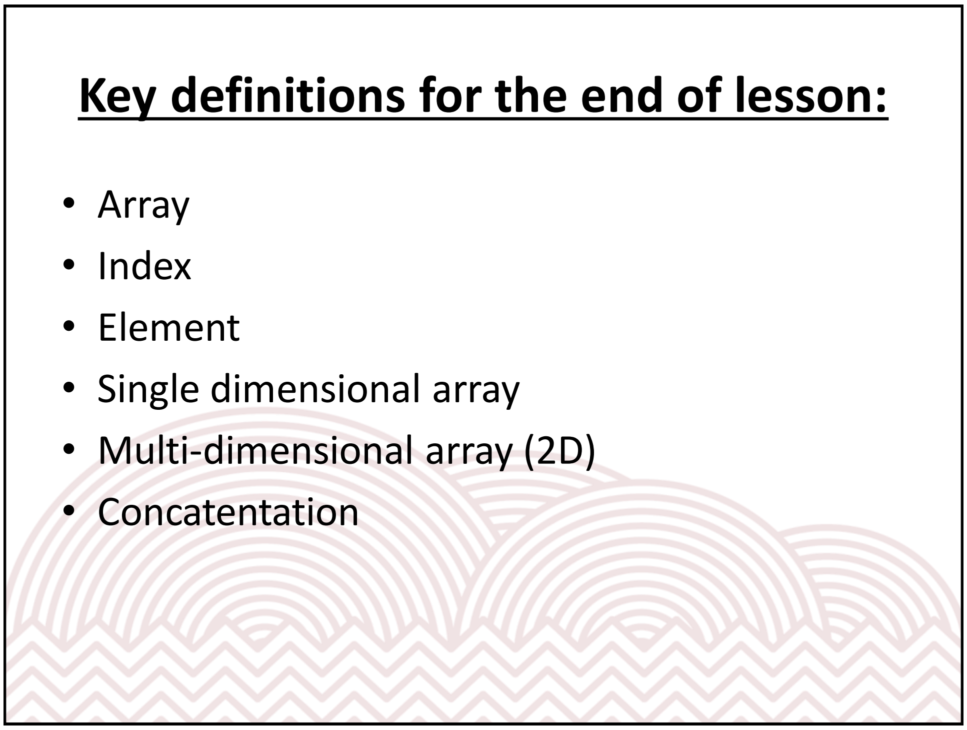


Chapter 4 – Arrays

By the end of this chapter you should:

- Understand how arrays are declared and used in C
- Be able to use arrays to hold lists of data in programs

1



Key definitions for the end of lesson:

- Array
- Index
- Element
- Single dimensional array
- Multi-dimensional array (2D)
- Concatentation

2

What is an array?

- An array is an alternative to a list
- It's a collection of variables that **all have the same name and data type** but each element of the array can be accessed using an **index** number e.g. name[1], name[2], name[3]
- They can be collections of integers, characters (a string) or floats etc - all the data in array will be of the same type
- To work with arrays you need to confident in your use of loops

3

Why use arrays?

- Let's say that you wanted to read in a list of ten numbers
- Using an array you can use a loop to enter the numbers into a list, read through all (or some of) the numbers stored and use the ***index*** to access an individual item in a list in fewer lines of code
- The loop counter is often used as the array index e.g. **i** (in the example on the right)
- You can also use loops really easily to initialise and set each memory location in the array to 0 e.g.
  You can also print out all items in an array in a similar way

```
int nums[10]

for (i=0; i<10; i++)
{
        nums[i] = 0;
}
```

4

Example 1- storing a list of numbers

```
int nums[10], guess=0, x=0;

for (x=0; x<10; x++)
{
    printf("Please enter number %d: ", x+1);
    scanf("%d", &nums[x] );
}
printf(“Which index would you like to check: ");
scanf("%d", &guess);

printf(“Index %d is: %d \n", guess, nums[guess] );
```

- **The array is declared just like a normal variable, but we have to state how many items will be in the array and put this in the square brackets**
- **Arrays start at index 0,** so this array will hold 10 numbers **- with the index numbers 0 - 9**
- This loop uses the **x** counter variable as the **index** so it can save data to e.g. nums[0], nums[1] etc
- **A number (guess) is then read in from the user**
- That number is used as the index number to access a particular element in the array and display the data stored in it on screen

5

Exercise 1

Create / edit the program on the last slide so that ten scores are stored and then a loop used to move through the array to add up the data and work out the average score.

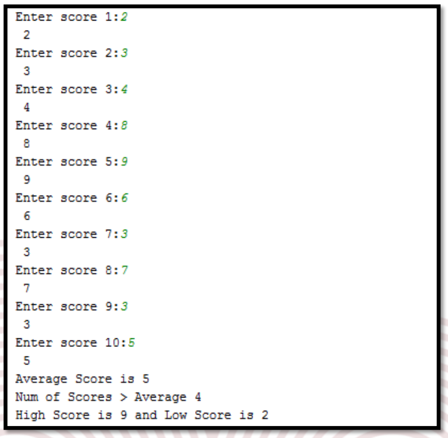
Requirements:

- The average score is output to the user
- The program outputs how many of the scores entered are higher than the average score
- The program outputs which is the lowest and which is the highest score

Example output on next page

6

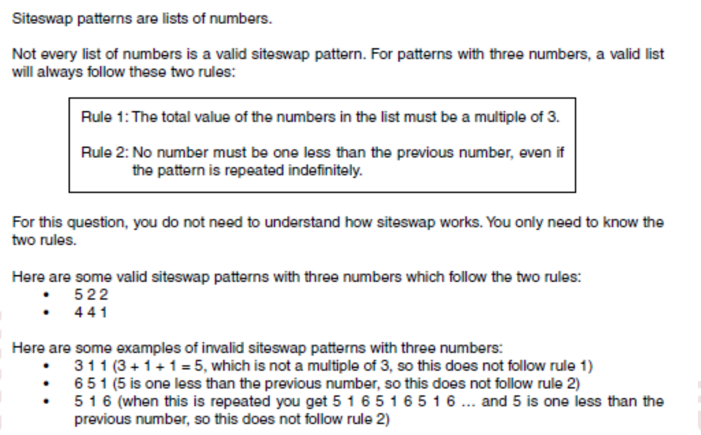
Example output:



```
Enter score 1:2
 2
Enter score 2:3
 3
Enter score 3:4
 4
Enter score 4:8
 8
Enter score 5:9
 9
Enter score 6:6
 6
Enter score 7:3
 3
Enter score 8:7
 7
Enter score 9:3
 3
Enter score 10:5
 5
Average Score is 5
Num of Scores > Average 4
High Score is 9 and Low Score is 2
```

7

Exercise 2



Siteswap patterns are lists of numbers.

Not every list of numbers is a valid siteswap pattern. For patterns with three numbers, a valid list will always follow these two rules:

Rule 1: The total value of the numbers in the list must be a multiple of 3.

Rule 2: No number must be one less than the previous number, even if the pattern is repeated indefinitely.

For this question, you do not need to understand how siteswap works. You only need to know the two rules.

Here are some valid siteswap patterns with three numbers which follow the two rules:
- 5 2 2
- 4 4 1

Here are some examples of invalid siteswap patterns with three numbers:
- 3 1 1 (3 + 1 + 1 = 5, which is not a multiple of 3, so this does not follow rule 1)
- 6 5 1 (5 is one less than the previous number, so this does not follow rule 2)
- 5 1 6 (when this is repeated you get 5 1 6 5 1 6 5 1 6 … and 5 is one less than the previous number, so this does not follow rule 2)

Write a program that tests for Siteswap patterns, it should allow the user to enter 3 numbers into an array and then output valid or invalid.

8

Intro to Strings in C

- In C, a word or string of words is stored as an **array of characters**
  - They are defined as a char data type
  - ...with bounds to define the max characters we want to store e.g. **char name[20];**
- Because a string is an array, that means we can either access individual parts of it, or just use it as a whole

The rules

- The [20] tells the compiler how many characters long your string may be at a maximum
- BUT an invisible **end of string marker** is added at the end of the text array (more later about this)
- SO you have to allow for enough space for the text to be entered **+1 for the end of string marker**. E.g. if you are storing a 6 letter word you need to declare an array with 7 indexes to store it properly.

9

Example 2 – how to extract a letter from a word:

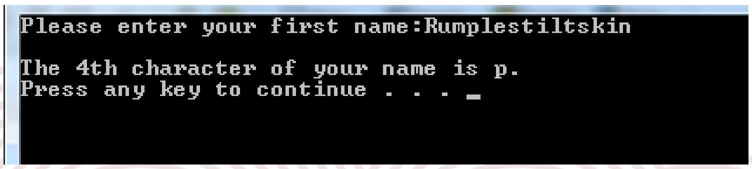
Creates an array for 19 characters (plus the marker)

Accesses the 4th element of the array

```
char firstName[20];

printf("Please enter your first name:");
gets(firstName);

printf("\nThe 4th character of your name is %c. \n", firstName[3]);
```



```
Please enter your first name:Rumplestiltskin

The 4th character of your name is p.
Press any key to continue . . . _
```

10

Strings are weird….

- Try running this program:
- Enter:
  - Your first name
  - Your whole name with a space between names
  - Your whole name with no space

ALSO: CLion auto outputs the string you just entered – which is really irritating – just ignore it!

```
int main() {
    char Name[10];

    printf("Enter Name: ");
    scanf("%s",Name);
    printf("Hello %s\n", Name);
    for(int i=0;i<10;i++){
        printf("%c",Name[i]);
    }
    printf("\n");
    for(int i=0;i<15;i++){
        printf("%c",Name[i]);
    }
    return 0;
}
```

11

Strings are weird….

- You hopefully saw:
  - It will output the string entered BUT it doesn't actually save any data past the number of characters you defined in the char array [10 characters]
  - Also, that a **scanf** stops reading data after a space is entered
  - AND a scanf for a %s DOESN'T use a pointer (as each letter is stored in a different memory location)
- Try running this version of the code (key parts in red have changed) and enter your full name with a space
  - You'll should see that the name outputs as normal in the first two loops, but in the last loop your name outputs plus other random characters
  - When you declare & enter a string, an **end of string marker** is used so a program knows when the data string ends
  - We can use the **gets()** function from the **string.h** library instead of a scanf when we want more than one word

```
#include <string.h>

int main() {
    char Name[100];

    printf("Enter Name: ");
    gets(Name);
    printf("Hello %s", Name);

    printf("\n");
    for(int i=0;i<100;i++){
        printf("%c",Name[i]);
    }
    return 0;
}
```

12

How to pre-fill an array

To pre-fill the array, you can enter data into each individual element like this:

```
nums[0] = 5;
nums[1] = 78;
nums[2] = 6;
nums[3] = 23;
```

Or you could loop through the array:

```
for (i=0; i<20; i++) {
    nums[i]= 0; }
```

Or you can initialise it when declaring the array e.g.

- **int nums[4] = { 7, 67, 8, 12};**
- **char stringData[7] = {"Dinner"}**
- **int myArray[4]={}** //this will initialise to null

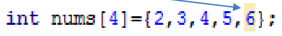
If you leave the brackets empty, the compiler will set the array size BUT this isn't good practice e.g.

- **int myArray[ ] = {0,0,0,0,0,0};**

WARNING

The IDE won't stop you entering more items of data than you have declared in the array size e.g**. int myArray[2]= {2, 4, 7, 8};**

If you do this it's likely to have a bit of a fit OR it will write over some other bit of memory so you have to be careful. The syntax checker should highlight items that are outside of the array bounds e.g. here the 6 is outside the array:



```
int nums[4]={2,3,4,5,6};
```

13

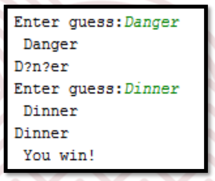
Exercise 2

Write a guess the word program that works as follows:

Requirements:

- The program should have a pre-set word, stored as a string
- The user should try to guess the word
- If a letter in the user's guess matches the letter in the same position in the pre-set word, the output string should show the letter contained at that position
- If a letter in the user's guess does NOT match the letter in that position in the pre-set word, it will display a question mark at that position.
- The user should get 5 chances to guess the correct word, after which the program should output "You Lose" and stop.

For example: if the correct work is "DINNER", and the guess is "DANGER", the procedure would output "D?N?ER"



```
Enter guess:Danger
 Danger
D?n?er
Enter guess:Dinner
 Dinner
Dinner
 You win!
```

14

Multi-dimensional arrays

So far, we've seen single dimensional arrays used to hold lists of data. However, you can have arrays with more than one index - a multi dimensional array.

You may only need to be able to use 2 dimensional arrays, which you can picture as a (sort of) table data structure.

They're declared like this: **int tableArray [10][2];**

The first set of indexes represents the number of 'rows' and the second index is for the columns.

This table would be created in memory:

tableArray	[0]	[1]
[0]		
[1]		
[2]		
[3]		
[4]		
[5]		
[6]		
[7]		
[8]		
[9]		

15

Example 2 – a 2D array

```
main()
{
int Grid[5][2] = {1, 10, 2, 20, 3, 30, 4, 40, 5, 50};
int x, y; //for the for loop counters

  for (y = 0; y < 5; y ++)    //controls the number of rows
  {
       for(x = 0; x < 2; x ++)    //controls the columns
       {
           printf("x%i y%i: %i\t", x, y, Grid[y][x]);
       }
     //At the end of a row, move down one space
     printf("\n");
  }
}
```

TASK 1
Copy the example into CLion and have a look at how it works.

Try changing it to a 10 by 10 array that displays 1 to 10 in the x column and 10 to 100 in the y column.

16

Extension Tasks

1. **Write a program that will read in a string of up to 80 characters and will then count the occurrences of vowels and consonants** (Use at least one loop to do this).

   At the end of the program a message should be output for the user to tell them how many of each were in the string they entered.

2. **Write a spelling program using an array of strings (like a list of words).**

   Requirements:
   - Your program should store at least 10 words.
   - When the program is run, the word should be displayed showing one letter missing.
   - The user should then be asked what the missing letter is.
   - A message should then be displayed telling the user if they are right or wrong
   - At the end of the program the user should be given a total showing the amount of correct answers they got

17

A/ A* Grade Extension Task

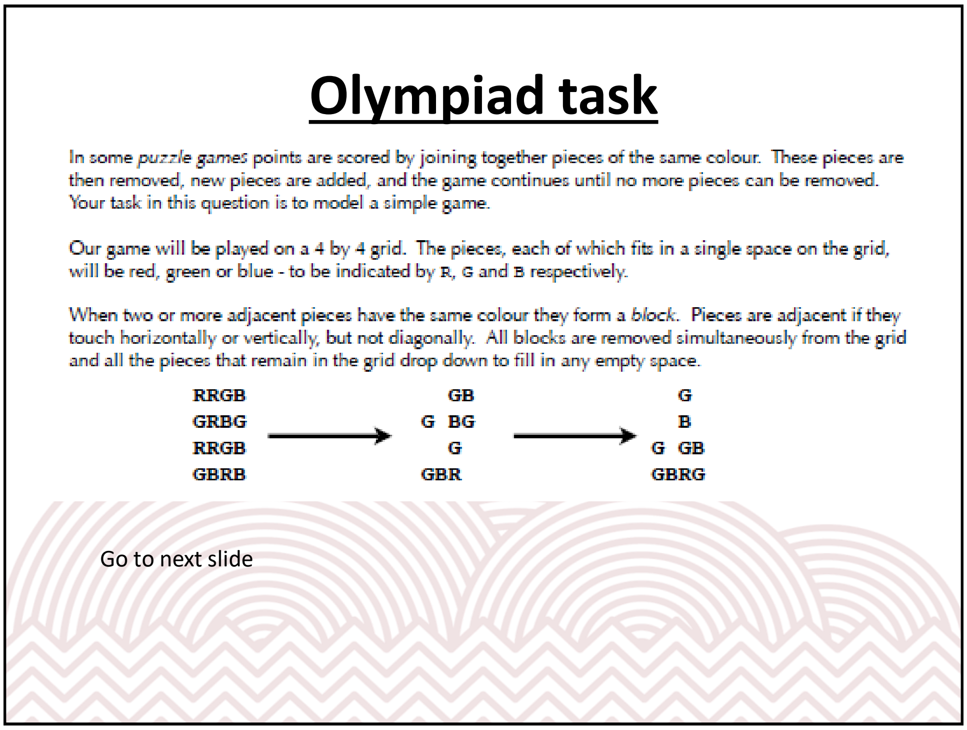
Write a sorting program that:

- **Uses a loop and the 'rand' function to generate 10 numbers and save them into an array**
- **Sorts the array of 10 integers into ascending order (smallest to largest)**
- **Show each stage of the sort, at the end of each full pass through the array**

  Use this link to remind yourself how to do a bubble sort (Scroll down to the bubble sort section) – and then implement one to solve this problem.

  http://www.cprogramming.com/tutorial/computersciencetheory/sorting1.html

18



Olympiad task

In some *puzzle games* points are scored by joining together pieces of the same colour. These pieces are then removed, new pieces are added, and the game continues until no more pieces can be removed. Your task in this question is to model a simple game.

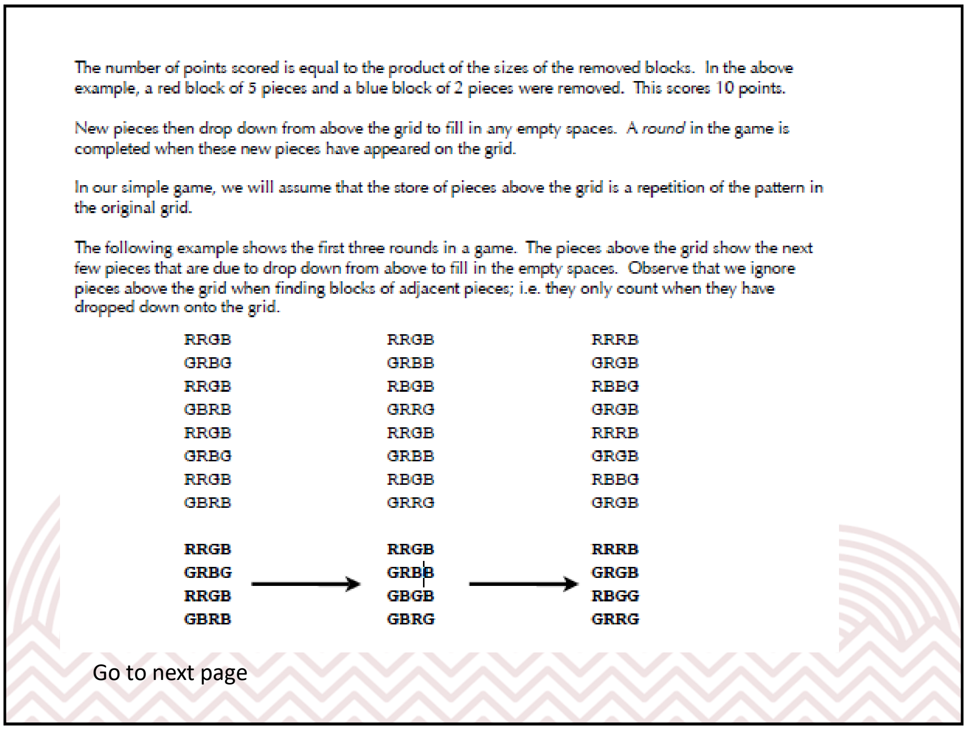
Our game will be played on a 4 by 4 grid. The pieces, each of which fits in a single space on the grid, will be red, green or blue - to be indicated by R, G and B respectively.

When two or more adjacent pieces have the same colour they form a *block*. Pieces are adjacent if they touch horizontally or vertically, but not diagonally. All blocks are removed simultaneously from the grid and all the pieces that remain in the grid drop down to fill in any empty space.

```
RRGB              GB               G
GRBG    ----->  G BG    ----->     B
RRGB              G              G GB
GBRB            GBR              GBRG
```

Go to next slide

19



The number of points scored is equal to the product of the sizes of the removed blocks. In the above example, a red block of 5 pieces and a blue block of 2 pieces were removed. This scores 10 points.

New pieces then drop down from above the grid to fill in any empty spaces. A *round* in the game is completed when these new pieces have appeared on the grid.

In our simple game, we will assume that the store of pieces above the grid is a repetition of the pattern in the original grid.

The following example shows the first three rounds in a game. The pieces above the grid show the next few pieces that are due to drop down from above to fill in the empty spaces. Observe that we ignore pieces above the grid when finding blocks of adjacent pieces; i.e. they only count when they have dropped down onto the grid.

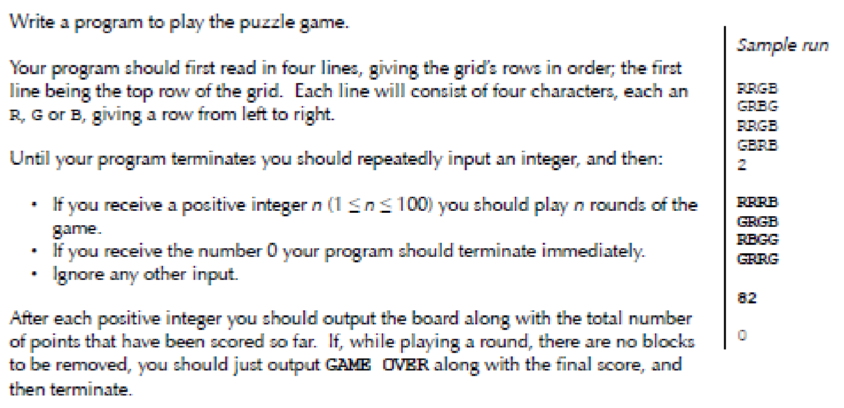
```
RRGB            RRGB            RRRB
GRBG            GRBB            GRGB
RRGB            RBGB            RBBG
GBRB            GRRG            GRGB
RRGB            RRGB            RRRB
GRBG            GRBB            GRGB
RRGB            RBGB            RBBG
GBRB            GRRG            GRGB

RRGB            RRGB            RRRB
GRBG  ----->    GRBB  ----->    GRGB
RRGB            GBGB            RBGG
GBRB            GBRG            GRRG
```

Go to next page

20

Now for the hard bit!



Write a program to play the puzzle game.

Your program should first read in four lines, giving the grid's rows in order; the first line being the top row of the grid. Each line will consist of four characters, each an `R`, `G` or `B`, giving a row from left to right.

Until your program terminates you should repeatedly input an integer, and then:

- If you receive a positive integer *n* ($1 \le n \le 100$) you should play *n* rounds of the game.
- If you receive the number 0 your program should terminate immediately.
- Ignore any other input.

After each positive integer you should output the board along with the total number of points that have been scored so far. If, while playing a round, there are no blocks to be removed, you should just output **`GAME OVER`** along with the final score, and then terminate.

Sample run

```
RRGB
GRBG
RRGB
GBRB
2

RRRB
GRGB
RBGG
GRRG

82

0
```