

Computer Programming Assignment #2 2003/4

You are to write a grading program for school use.

Your program should accept and store students names and marks. Once the marks have been entered the program should then print those details out again but now with the FETAC grade included.

Sample output from a run of a possible implementation is shown here:

```
[fachtna@office cprogs]$ gcc a2-2004.c
[fachtna@office cprogs]$ ./a.out
This program is designed to accept students names and marks and
to grade them according to the FETAC marking scheme. Up to 10
names may be entered.
```

```
Please enter the number of students to process: 4
```

```
=====
Enter students name and mark: Andrew 23
Enter students name and mark: Willy -9
The mark for Willy is not valid. Please re-enter: 90
Enter students name and mark: James 76
Enter students name and mark: Peter 54
=====
```

```
Name Mark Grade
Andrew 23 F
Willy 90 D
James 76 M
Peter 54 P
[fachtna@office cprogs]$
```

You are to try to present your program output attractively. If it is possible to do so you should also try to include a sanity check on the input data as, for example, negative marks aren't allowed.

You may assume that there will be a maximum of ten names of up to ten characters each.

Submission Date: Friday 7th May 2004, 15:15

Submission Method: Electronic & Paper

Submission Includes: Signed Cover Sheet, Source Code, Flow Charts, Screen Shots/Sample Data, Any other relevant materials

Arrays

An array is a collection of variables that share a name. Each variable is identified by a combination of the shared name and a number. A numeric array of 5 elements is declared in this manner:

```
int some_array[5];
```

When using the array elements at a later stage the same method for reading and writing that *type* of data is followed, just add the number of the specific element you wish to work with. These samples show how to read the first and write the second elements of the array:

```
scanf ("%d", &some_array[0]);  
printf ("%d", some_array[1]);
```

Note that the first element is zero, not one. The 5 elements are 0,1,2,3,4 – if you want to use 1,2,3,4,5 you can declare an array of 6 elements and not use the first. The element number is often called the array subscript.

Using Arrays

The reason that arrays are used is to allow us to program to carry out some action on the whole collection of elements at once. Typically this involves the use of a variable to specify the array subscript. This is because we can use a **while** loop to ensure that a range of values in turn go through the subscript variable. It is a relatively simple thing to generate a sequence of 1,2,3,4,5 using a loop:

```
subscript_variable = 1;  
while (subscript_variable <= 5)  
{  
printf ("%d\n", some_array[subscript_variable]);  
subscript_variable++;  
}
```

Character Arrays

A character is just one letter or symbol from the keyboard. Several characters combined is called a string. These are created using a character array.

Words and names are stored in character arrays. A character array is declared in this form:

```
char sample_char_array[20];
```

When reading or writing such an array the **string** or **%s** marker is used in the I/O statement:

```
printf ("%s", sample_char_array);
```

An Array Of Arrays

An array of character arrays (strings) is declared as follows

```
char string_array[5][20];
```

This is actually a two dimensional array of characters. You can think of it as a 5 by 20 grid. This is similar to having storage for 20 words of 5 characters each. To use each word just use a single subscript with the string (**%s**) formatter:

```
scanf ("%s", &string_array[1]);
```

if Statements

The FETAC marking scheme is as follows:

```
0-49 Unsuccessful  
50-64 Pass  
65-79 Merit  
80-100 Distinction
```

Marks below 0 and above 100 are not possible and therefore are not valid input.

To decide which grade is allocated an **if** statement is used. The general form of an **if** statement is:

```
if (condition)
```

```
statement1;
```

If the (condition) is true then statement1 will be carried out. Otherwise it will not.

else Statements

Where there is only one other possibility – when the (condition) evaluates as false an else may be used:

```
if (condition)
statement1;
else
statement2;
```

Nested if/else Statements

Note that statement1 and statement2 could both be other if/else combinations or multiple statements enclosed in curly brackets:

```
if (condition1)
{
statement1;
statement2;
}
else
if (condition2)
statement3;
else
statement4;
```

Logical Operators

Logical operators are used where there is more than one condition to be evaluated. In this circumstance you may wish for the combination of the conditions to all be true (AND) or that only one or more be true (OR). These are the same logical operators used in spreadsheets, just written differently. AND is written as && and OR is written as ||

```
if ( (condition1) && (condition2) )
statement1; // both conditions are true
else
if ( (condition1) || (condition2) )
statement2; // one or other of the conditions is true
```

Logical operators can be used in **while** statements as well:

```
while ( (condition1) && (condition2) )
{
body of loop goes here;
}
```

Important Note

In these pages care has been taken to ensure that all brackets, semi-colons and punctuation marks are in the correct place. When applying the general forms of arrays, if/else statements etc make sure that the correct numbers and positionings of these in your implementation or compilation and run-time errors will occur.

Suggested Implementation

This is the suggested set of steps to complete the program. It is assumed that each step when completed will compile and run and is tested before moving on. Each step builds on the earlier steps.

Full marks will be available for completion of all steps but marks will be awarded for any

steps completed and submitted as long as the submitted code will compile. Marks are also awarded for indentation, commenting, use of constants, meaningful variable names, elegance etc.

Write a program that:

1. Reads and writes back out one letter
2. Reads and writes back out one name
3. Reads and writes back out one name and mark
4. Declare an array of names
5. Declare an array of numbers
6. Modify program from **3** to use the array variables instead of the variables originally used
7. Encapsulate **6** in a **while** loop
8. Remove the writing back out aspect of **7** and place in a new loop after the first
9. Ensure that the loops at **8** are able to accept, store and display more than one name, mark pair
At this point you should have a program that reads in names and marks and then prints them out in the same order with no other processing being done
10. In the second loop introduce an **if** and **else** statement that determines whether the student has passed or failed
11. Add further **elses** and **ifs** to account for the other possible grades
12. Present the output attractively. Use **\t** for a **tab** character in your output
At this point your program should write out the names, marks and grades with no data validation being done
13. Use a **while** loop within the first **while** loop to repeatedly request valid numeric input. A logical (**&&**, **||**) in the **while** loop may make this easier
14. Add output formatting of your choice to make the screen display attractive
15. Ensure your program is properly indented and commented before submitting.