



UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

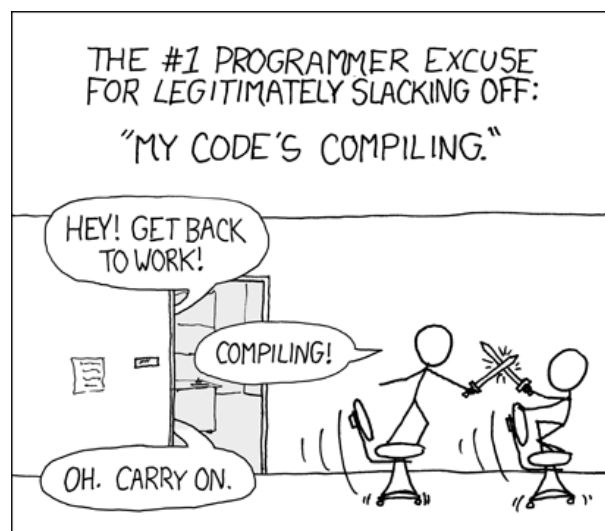
Computer Programming 143

Practical 5

2016

Aim of Practical 5:

- Declare an array and load values (including random values)
- Access array entries using a **for**-loop
- Manipulate an array
- Write a function that uses recursion



Instructions

1. Attendance is **compulsory** for all the practical sessions of your assigned group. See the study guide for more details.
2. The last section (usually the last 30 minutes) of the practical will be used for a test.
3. If more than three tests have been missed for what ever reason, you will receive an **INCOMPLETE** for the subject. See the study guide for more details.
4. You must do all assignments **on your own**. Students are encouraged to help each other **understand** the problems and solutions, but each should write his/her own code. By simply copying someone else's code or solutions, you will not build an understanding of the work.
5. You are responsible for your own progress. Ensure that you understand the practical work. Check your work against the memorandum that will be posted on Wednesday afternoons on learn.sun.ac.za.
6. Use H:\CP143 as the Eclipse workspace folder for all projects but it is highly suggested that you use a **flash drive to backup** all work done.
7. Create a new project **for each assignment**. See *Creating an Eclipse Project* in the Basic Eclipse Project Handling Notes for instructions on how to do just that.
8. Include a comment block at the top of each source file according to the format given. It must include the correct filename and date, your name and student number, the copying declaration, and the title of the source file.
9. **Indent your code correctly!** Making your code readable is not beautification, it is a time and life saving habit. Adhere to the standards (refer to the documents on SUNLearn). You can use Ctrl+A and then Ctrl+I to auto-indent.
10. Comment your code sufficiently well. It is required for you and others to understand what you have done.

Question A

Goal: *Perform operations on a numerical array.*

Getting started

1. Create a project named Assignment5A. Make sure that this is the only open project in the workspace before compiling the program.
2. Include the **standard comment block** above your main function. Also, comment your whole program appropriately.
3. Read the complete question before you start programming.
4. Either draw a **program flowchart** or write the **pseudocode before** you start programming. A demi/instructor may ask to see this when they come to assist you.

Program description

1. In the main() function declare one array of type **int** with 10 elements. You will be doing the following operations on this array - assign values to it, display it, do a mathematical operation on it and once again display it, this time in reverse.
2. Assign values to the array:
 - (a) Use the library function rand() to generate pseudo-random integers to populate the array with numbers in the inclusive range of 1 to 20 (mathematically shown as [1,20]) . The rand() function returns a value between 0 (inclusive) and RAND_MAX (exclusive). The value of RAND_MAX is set by the compiler, but is at least 32767. If we wish to change the range of possible numbers to something other than [0,RAND_MAX), we do the following calculation:

```
NewRandomValue = (rand() % ( upper - lower + 1 )) + lower
```

where lower and upper are the inclusive boundaries of the range of desired values, or put another way: values will now be chosen in the range [lower, upper].

- (b) Examples of implementing different ranges:
if the chosen range is [1,100] then the code will be:

```
NewRandVal = (rand() % (100 - 1 + 1)) + 1;  
or  
NewRandVal = (rand() % 100) + 1;
```

if the chosen range is [0,9] then the code will be:

```
NewRandVal = (rand() % (9 - 0 + 1)) + 0;  
or  
NewRandVal = (rand() % 10);
```

- (c) Before using the `rand()` function you should seed it with a value. The most common is with time. We use the function `srand(time(NULL))` to seed the random generator. The function `time` returns a value that the function `srand` uses to seed the random number generator. To use `time` include the library `<time.h>` To see the effect this has, finish the next point of displaying the array values, then do the following: Comment the call to `srand` out. Run your program a number of times and note the values that are generated and stored in the array. Now put the call to `srand` back. Run it again a couple of times and once again note the values that are generated each time. Makes a difference, doesn't it?
3. Display the array:
Step through the array and display the index of the array element as well as the value of the array element.
 4. Do a mathematical operation on the array:
Step through the array and multiply each element with 100. Note that doing operations like these must be done element by element – you can't say `arrName = arrName * 100`.
 5. Display the array a second time:
Once again display the index and values of the array, but this time in **reverse** order according to its index, i.e. display the last element first and the first element last.

Submission instructions

1. Ensure that your code is indented correctly and that the `{ }` braces are on the correct lines. Use the prescribed textbook as guideline.
2. Ensure that you copy the **Assignment5A** project folder to a flash drive as a backup.

Question B

Goal: *Perform operations on a character array with functions.*

Getting started

1. Create a project named Assignment5B. Make sure that this is the only open project in the workspace before compiling the program.
2. Include the **standard comment block** above your main function. Also, comment your whole program appropriately.
3. Read the complete question before you start programming.
4. Either draw a **program flowchart** or write the **pseudocode before** you start programming. A demi/instructor may ask to see this when they come to assist you.

Program description

1. In the `main()` function declare an array of type `char` with 100 elements. You will be doing the following operations on this array - assign values to it, display it, do a sort on it and once again display it - all with functions. These functions, of which there are three, are described in the following points. You will call all of these functions from `main()`.
2. All of the functions will be called with the array and the size of the array as arguments. The prototype of each of these functions will be according to the following template:

```
void functionName(char arrayName[], int sizeOfArray);
```

3. Assign values to the array:
 - (a) Write a function that receives a character array by reference and its size by value. The purpose of this function will be to assign the lower-case letters, from 'a' to 'z', in a random fashion to this character array that it has received through its parameter list. The easiest way to generate random letters is with `rand()` in the same way that you generated random integers in the previous question. Though `rand()` returns integers, we can use it to generate letters by making the range of possible values [97, 122] and by casting `int` to `char`.
 - (b) Why is the chosen range [97,122]? These are the numbers that represent 'a' to 'z' in ASCII code. The ASCII standard assigns a number to most alpha-numerical characters. For example the character for 'l' is represented by the number 108 in this code, 'a' is 97, 'b' is 98 and so forth, while 'A' is 65, 'B' is 66 and so forth.
 - (c) The value you will get from `rand()` and the % and + operations you do on it results in an integer value. You need to assign this integer value to your array element, but your array is of type `char`. The correct way to do it is to first cast the value to `char` and then do the assignment, as in the following example:

```

char c;
int i;
i = 97; //the numerical value 97 is assigned to i
c = (char)i; // the numerical 97 is cast to a character ->
             // it is not the number 97 any more,
             // but the letter 'a' (according to ASCII)

```

- (d) If you do not cast the numerical value to char, your code will probably still work - automatic casting has then taken place. The problem with relying on automatic casting is that it might not happen for all the computer systems you might work on. It is a very good coding habit to always use explicit casting.
- (e) Remember to seed the random number generator!

4. Display the array:

Write a function that receives a character array by reference and its size by value. The purpose of this function would simply be to step through the array and display the index of the array element, as well as the value of the array element.

5. Sort the array:

Write a function that receives a character array by reference and its size by value. The purpose of this function would be to sort the array using bubble sort. You can refer to your textbook/lecture slides on how to code bubble sort or you can follow either one of the following pseudocode examples. (The algorithm is exactly the same for the two pseudocode examples. The only reason why it is given twice is to show you that different wording is acceptable in pseudocode.)

Bubble sort - pseudocode example 1:

```

/*
 * begin function bubbleSort(reference to array b, size)
 *   for pass from 1 to (size - 1) in increments of 1
 *     for i from 0 to (size - 2) in increments of 1
 *       if (element i of b) > (element (i+1) of b)
 *         //swap the the values in these two elements
 *         hold = element i of b
 *         element i of b = element (i+1) of b
 *         element (i+1) of b = hold
 *   end function
 */

```

Bubble sort - pseudocode example 2:

```

/*
 * begin bubbleSort(reference to array b,size)
 *   for pass in range [1, size) in increments of 1
 *     for i in range [0, (size - 1)) in increments of 1
 *       if b[i] > b[i+1]

```

```
*           //swap the the values in these two elements
*           hold = b[i]
*           b[i] = b[i+1]
*           b[i+1] = hold
* end
*/
```

6. Display the array a second time:

To display the array again, do NOT write another function - just call the function that you have already written in point 4.

Something to think about: The American Standard Code for Information Interchange, better known as the ASCII code is used worldwide to represent characters in numerical format. Computers can only “understand” numbers. Everything that happens on your computer, from the amazing graphics of your latest game, or the assignments you write for your degree to the music you listen to, are only numbers to the computer.

Submission instructions

1. Ensure that your code is indented correctly and that the {} braces are on the correct lines. Use the prescribed textbook as guideline.
2. Ensure that you copy the **Assignment5B** project folder to a flash drive as a backup.

Question C

Goal: Write two recursive functions.

Getting started

1. Create a project named Assignment5C.
2. Include the **standard comment block** above your main function. Also, comment your whole program appropriately.
3. Read the complete question before you start programming.
4. Either draw a **program flowchart** or write the **pseudocode before** you start programming. A demi/instructor may ask to see this when they come to assist you.

Program Description

1. For this question you will write two recursive functions and the `main()` that will call them. The first recursive function will simply print a counter to the screen. The second one will find the greatest common divisor between two given integers.

- (a) The “print a counter” function:

Write a recursive function that takes no arguments and returns nothing. It should print a counter value to the screen. It must be the recursive equivalent of the following **for** loop:

```
int x;  
for (x = 0; x < 10; x++) {  
    printf("\n%d", x);  
}
```

Hint: make use of a **static** variable.

The function prints the counter, increments the counter and then calls itself again if the counter has not reached the desired value.

- (b) The GCD (greatest common divisor) function:

Given two integers, find the biggest integer that divides evenly into both – in other words, find the greatest common divisor (GCD). For example: the GCD of 20 and 10 is 10, the GCD of 99 and 100 is 1, and the GCD of 12 and 42 is 6.

We will calculate the GCD recursively using **Euclid's algorithm**. This algorithm is built on the following property of the GCD:

$$\text{gcd}(x, y) = \begin{cases} x & \text{if } y = 0 \\ \text{gcd}(y, x \bmod y) & \text{if } y \neq 0 \end{cases}$$

Notice that this definition of the GCD operator includes the operator itself on the right-hand side – in other words, it is defined **recursively**.

We will create a recursive function called `gcd(int x, int y)`, which will implement this equation. If the second number is zero, it will simply return the first number; otherwise, it will call itself recursively.

For example, the GCD of 180 and 24 can be calculated as follows:

$$\begin{aligned}\gcd(180, 24) &= \gcd(24, 180 \bmod 24) = \gcd(24, 12) \\ \gcd(24, 12) &= \gcd(12, 24 \bmod 12) = \gcd(12, 0) \\ \gcd(12, 0) &= 12\end{aligned}$$

(c) The calling function:

Your `main()` function will be the calling function for the other two. Ensure that you only have the function prototypes above the `main()` function. Call the “print a counter” function first. Call and print the results of the “greatest common divisor” function with the following pairs of values: 10 and 20; 20 and 10; 777 and 99; 345 and 6000.

Face the world: In 1844 it was proven that, if the smallest number has n digits, Euclid’s algorithm never takes more than $5n$ recursion steps to calculate the GCD. This means that the time a computer will take to find a GCD is a *logarithmic* function of the size of the numbers.

This was one of the first results in analysing the *computational complexity* of algorithms, which is the focus of a lot of the work done in computer science. You may have different algorithms that can perform the same function, but their running times may depend differently on the amount of input data. The best algorithms are the ones that run in *constant time* – their running time is independent of the amount of data. Then you get, in increasing order of complexity, algorithms that run in *logarithmic time*, *linear time*, *polynomial time* (e.g. quadratic), and *exponential time*. If you have a large amount of data, the difference in running time between a linear algorithm and a logarithmic one can be huge.

There will often be one possible algorithm which is very easy to implement, but is computationally inefficient, while another algorithm will be more efficient but harder to implement. The different sorting algorithms are good examples of this – there are simple but inefficient algorithms such as Bubble Sort, and more efficient ones such as Quick Sort that are harder to implement.

Submission instructions

1. Ensure that your code is indented correctly and that the `{ }` braces are on the correct lines. Use the prescribed textbook as guideline.
2. Ensure that you copy the **Assignment5C** project folder to a flash drive as a backup.