THE IMAGINATION UNIVERSITY PROGRAMME

RVfpgaEL2 Lab 1
C Programming

1. Introduction

Most computer programs are written in a high-level language such as C. This lab shows you how to create a C project in Catapult that you can run on the RVfpgaEL2 System. We first provide a tutorial on how to create and run a C program. Then we describe exercises for you to practice writing your own C programs.

IMPORTANT: Before starting RVfpgaEL2 Labs, you must have already completed the RVfpgaEL2 Getting Started Guide provided by the Imagination University Programme (https://university.imgtec.com/). For example, if you have not already, install the software tools (at least Catapult) following the instructions in the RVfpgaEL2 Getting Started Guide. Also, make sure that you have copied the ***RVfpga*** folder that you downloaded from Imagination's University Programme to your machine. We will refer to the absolute path of the directory where you place folder RVfpgaEL2_Boolean as [*RVfpgaBooleanPath*]. The [*RVfpgaBooleanPath*]/src folder contains the Verilog and SystemVerilog sources for the RVfpga System, the RISC-V SoC that we will use and modify throughout the labs. The [*RVfpgaBooleanPath*]/Labs folder contains some programs that you will use during the labs.

2. C Program for RVfpgaEL2

You will complete the following steps to create and run a C program on RVfpgaEL2-Boolean and RVfpgaEL2-ViDBo using Catapult:

1. Create an RVfpgaEL2 project
2. Write a C program
3. Download RVfpgaEL2-Boolean onto the Boolean Board and compile, download, and run a C program on RVfpgaEL2-Boolean
4. Compile, download, and run a C program on RVfpgaEL2-ViDBo

PlatformIO: If you use PlatformIO instead of Catapult, you can find the instructions for creating a project in Lab 01 of RVfpga v2.2 (the RVfpga course targeted to VeeR EH1).

Step 1. Create an RVfpgaEL2 project

In folder *[RVfpgaBooleanPath]/Labs/Lab01*, create a directory called *Project1*.

In this course all the examples use the **cmake** build system. In this system, the build is defined in a *CMakeLists.txt* file. In *[RVfpgaBooleanPath]/Labs/Lab01/ProjectSources* we provide a baseline *CMakeLists.txt* file that we will use in this first example. Analyse this file, which is briefly described next:

- It first defines a C project called Test (projects in assembly will define it as ***ASM***, and projects combining C and assembly will use ***C ASM***):
```
project(Test C)
```

- It then selects the sources for the project, which in this case are in a single file:
```
${CMAKE_CURRENT_SOURCE_DIR}/src/Test.c
```

- It then assigns a `TARGET_NAME` to the executable, in this case, ***Test.elf***:
```
set(TARGET_NAME Test.elf)
```

- It then adds an executable target to be built from the source files listed above:

```
add_executable(${TARGET_NAME} ${SOURCE})
```

- Finally, it includes a common *cmake* file provided at: *[RVfpgaBooleanPath]/common*

```
include(${CMAKE_CURRENT_SOURCE_DIR}/../../../common/Common.cmake)
```

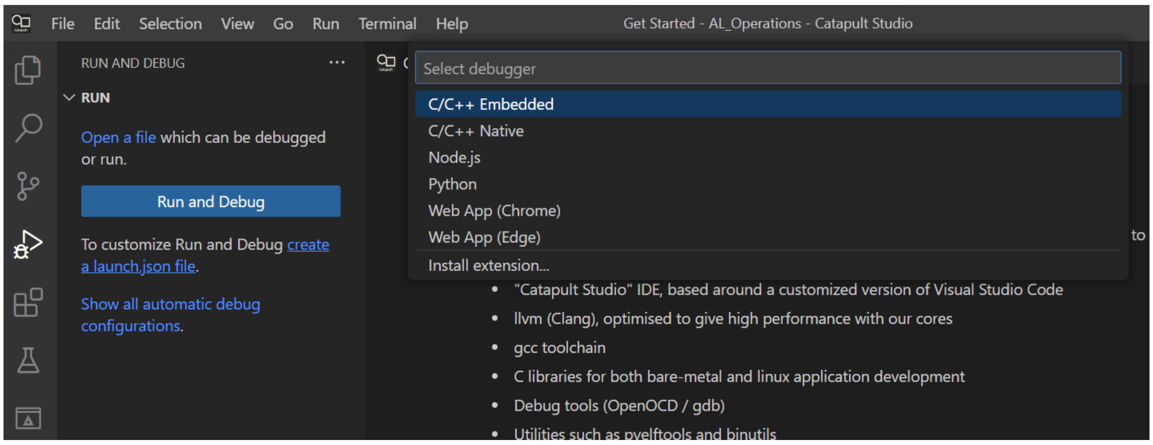
  Note that you may need to modify this path depending on the location of your project. You will see that some of the projects provided, such as the one we are analysing here, use the common folder located at *[RVfpgaBooleanPath]/common*, whereas other projects use a specific folder located inside the project folder itself (for example: *[RVfpgaBooleanPath]/Labs/Lab04/ImageProcessing/common_tuned/*). The reason for this is that some programs need to perform changes to the information included in the *common* folder.

Copy the provided *CMakeLists.txt* file into directory *[RVfpgaBooleanPath]/Labs/Lab01/Project1*.

Open Catapult. On the top menu bar, click on *File → Open Folder,* browse into directory *[RVfpgaBooleanPath]/Labs/Lab01*, select directory *Project1* (do not open it, but just select it) and click OK at the top of the window.

Then, if Catapult asks you for the kit for your project, select “Catapult SDK gcc-elf”.

Finally, as explained in the GSG, create files *tasks.json* and *launch.json* by clicking on the Run/Debug tab on the left hand side, then clicking “create a launch.json file”, and finally selecting “C/C++ Embedded” as the debugger type:


File Edit Selection View Go Run Terminal Help
Get Started - AL_Operations - Catapult Studio
RUN AND DEBUG
RUN
Open a file which can be debugged or run.
Run and Debug
To customize Run and Debug create a launch.json file.
Show all automatic debug configurations.
Select debugger
C/C++ Embedded
C/C++ Native
Node.js
Python
Web App (Chrome)
Web App (Edge)
Install extension...
• "Catapult Studio" IDE, based around a customized version of Visual Studio Code
• llvm (Clang), optimised to give high performance with our cores
• gcc toolchain
• C libraries for both bare-metal and linux application development
• Debug tools (OpenOCD / gdb)
• Utilities such as pyelftools and binutils

Step 2. Write a C program

Now you will write a C program. Start by creating directory **src** in folder *[RVfpgaBooleanPath]/Labs/Lab01/Project1*.

Then, click on File → New File (see Figure 1)

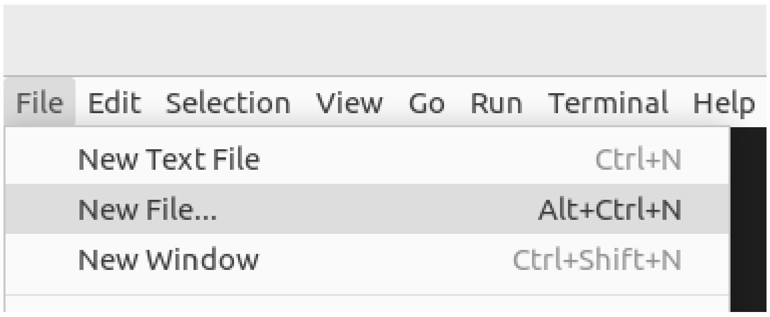


Figure 1. Add file to project

A menu window will open (see Figure 2). Select "Text File".

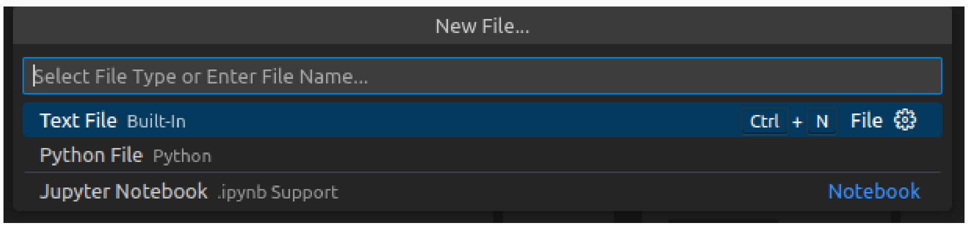
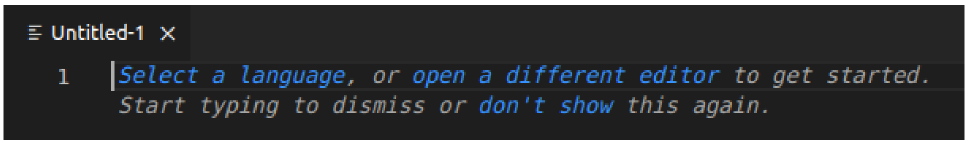

New File...
Select File Type or Enter File Name...
Text File Built-In
Ctrl + N
File
Python File Python
Jupyter Notebook .ipynb Support
Notebook

Figure 2. Select "Text File"

A blank file will open where you will be asked some questions (see Figure 3).



```
Select a language, or open a different editor to get started.
Start typing to dismiss or don't show this again.
```

Figure 3. Blank file

Click on "Select a language" and, in this case, choose C (see Figure 4).

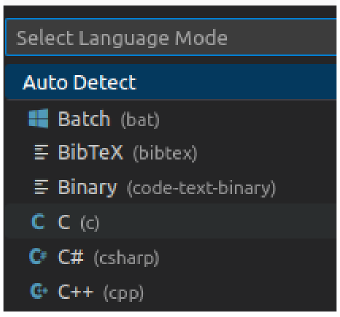


Figure 4. Select a language – C

Type (or copy/paste) the following C program into that window (see Figure 5). This program displays the value of the switches on the LEDs.

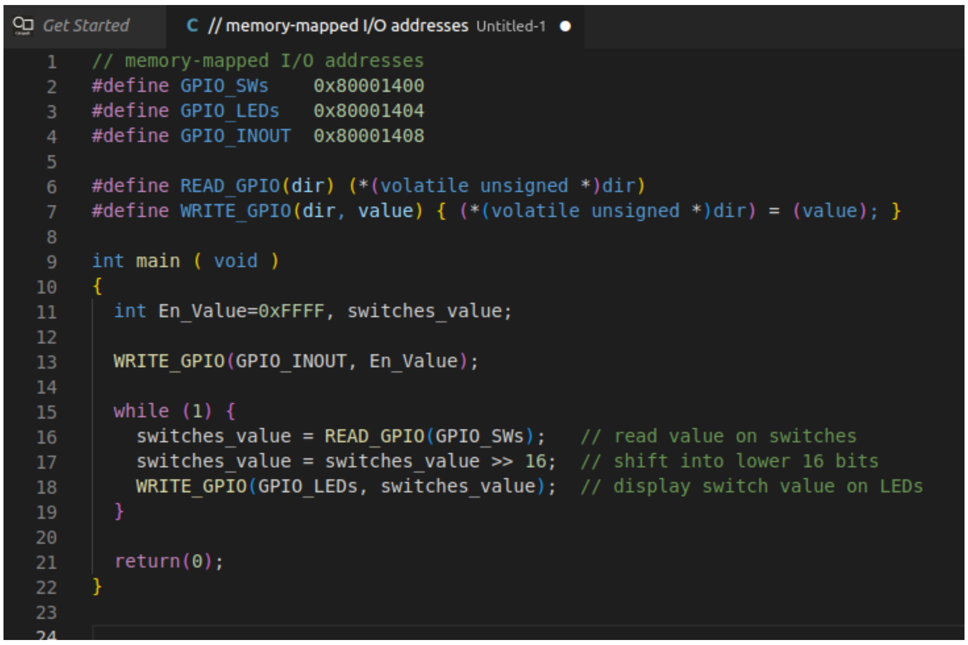
```
// memory-mapped I/O addresses
#define GPIO_SWs    0x80001400
#define GPIO_LEDs   0x80001404
#define GPIO_INOUT  0x80001408
```

```
#define READ_GPIO(dir) (*(volatile unsigned *)dir)
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }

int main ( void )
{
  int En_Value=0xFFFF, switches_value;
  WRITE_GPIO(GPIO_INOUT, En_Value);
  while (1) {
    switches_value = READ_GPIO(GPIO_SWs);   // read value on switches
    switches_value = switches_value >> 16;  // shift into lower 16 bits
    WRITE_GPIO(GPIO_LEDs, switches_value);  // display switch value on LEDs
  }
  return(0);
}
```

This program is also available in the following file for your convenience:

[RVfpgaBooleanPath]/Labs/Lab01/ProjectSources/DisplaySwitches.c



```
// memory-mapped I/O addresses
#define GPIO_SWs    0x80001400
#define GPIO_LEDs   0x80001404
#define GPIO_INOUT  0x80001408

#define READ_GPIO(dir) (*(volatile unsigned *)dir)
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }

int main ( void )
{
  int En_Value=0xFFFF, switches_value;

  WRITE_GPIO(GPIO_INOUT, En_Value);

  while (1) {
    switches_value = READ_GPIO(GPIO_SWs);   // read value on switches
    switches_value = switches_value >> 16;  // shift into lower 16 bits
    WRITE_GPIO(GPIO_LEDs, switches_value);  // display switch value on LEDs
  }

  return(0);
}
```

Figure 5. Enter C program

After entering the program into the pane, press Ctrl-s to save the file. Name it Test.c and save it in the `src` folder of the `Project1` directory (see Figure 6).

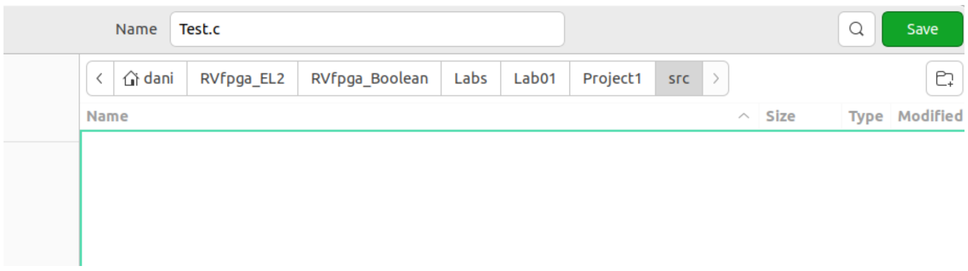


Figure 6. Save file as Test.c

This program first defines the addresses of the memory-mapped I/O registers connected to the LEDs and switches on the Boolean FPGA board using the following lines:

```
#define GPIO_SWs     0x80001400
#define GPIO_LEDs    0x80001404
#define GPIO_INOUT   0x80001408
```

The value of the switches is found by reading the register mapped to address 0x80001400 and values are displayed on the LEDs by writing the register mapped to address 0x80001404. The switch values are in the upper half of the register, and the LEDs in the lower half.

The GPIO_INOUT register defines whether a bit of the general-purpose I/O (GPIO) is an input or an output. The least significant 16 GPIO pins, 15:0, are connected to the 16 LEDs on the Boolean board. The most significant 16 GPIO pins, 31:16, are for the 16 on-board switches. A 0 indicates an input and a 1 indicates an output. So, the GPIO_INOUT register is written with 0xFFFF so that the switches are inputs to RVfpgaEL2-Boolean and the LEDs are outputs driven by RVfpgaEL2-Boolean.

Figure 7 shows the physical locations of the LEDs and switches on the Boolean FPGA board as well as the USB connector, ON switch, pushbuttons, and 7-segment displays.

Note that in Lab 6 we describe the GPIO features and RVfpgaEL2-Boolean GPIO hardware in detail. We also discuss how to use the other board peripherals, such as pushbuttons and 7-segment displays, in later labs (Labs 6-10).

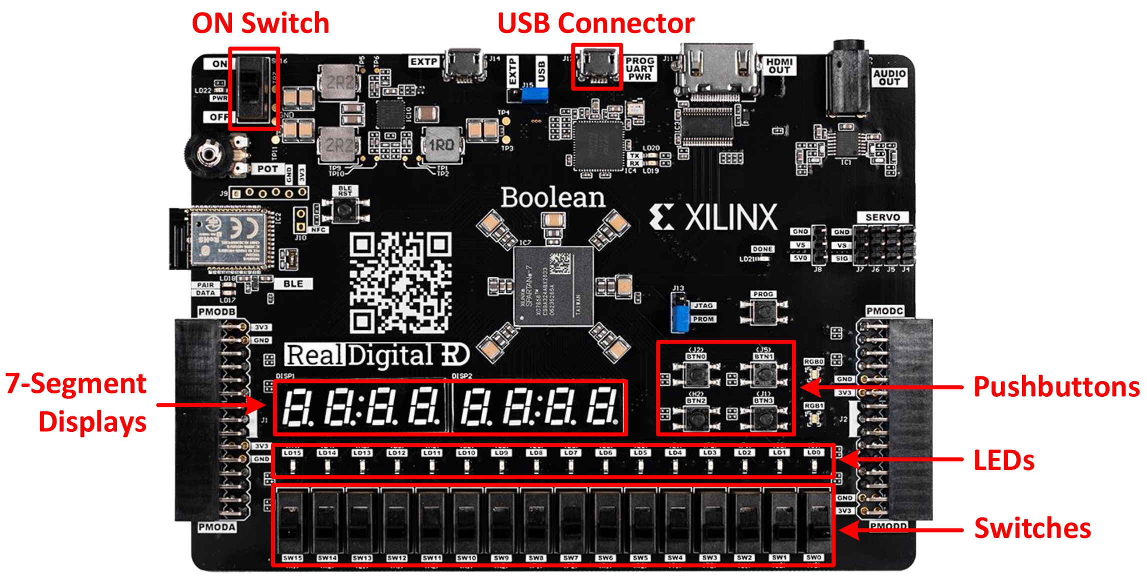


Figure 7. Real Digital's Boolean FPGA board's I/O interfaces
(figure of board from Real Digital)

After defining the memory-mapped I/O addresses of the LEDs and switches, the program does the following:

1. Defines the most significant 16 GPIO pins (which are connected to the switches) as inputs by setting the upper half of the GPIO_INOUT register to 0s and defines the least significant

16 GPIO pins (which are connected to the LEDs) as outputs by setting the lower half of the GPIO_INOUT register to 1s by executing the following code:

```
int En_Value=0xFFFF;

WRITE_GPIO(GPIO_INOUT, En_Value);
```

2. Repeatedly reads the value of the switches and writes that value to the LEDs by executing the code below. Recall that the value of the switches is read into the upper half of the memory-mapped I/O register, so the value must be shifted to the right by 16 bits before writing it to the memory-mapped I/O register physically connected to the LEDs.

```
while (1) {
  switches_value = READ_GPIO(GPIO_SWs);    // read value on switches
  switches_value = switches_value >> 16;  // shift into lower 16 bits
  WRITE_GPIO(GPIO_LEDs, switches_value);  // display switch value on LEDs
}
```

The READ_GPIO and WRITE_GPIO macros respectively read or write a value at the specified memory-mapped I/O address.

Step 3. Download RVfpgaEL2-Boolean onto the Boolean Board and compile, download, and run a C program on RVfpgaEL2-Boolean

You will now run the example program on RVfpgaEL2-Boolean. For that purpose, start by configuring and building the project as explained in detail in the Getting Started Guide and summarized next:

- Select kit “Catapult SDK gcc-elf” for the project.
- Select Debug as the build configuration.
- Click the Platform selection on the bar at the bottom of the screen and select “RVfpga Boolean via OpenOCD” for the platform.
- Clean the build and re-start it from scratch by selecting the CMake tab and selecting “Clean reconfigure all projects” from the “…” menu.
- Finally, build the program. You will see the build output in the built-in terminal within Catapult Studio. If successful, it should end with “Build finished with exit code 0”.

Download RVfpgaEL2-Boolean onto the board by clicking on the Platform selector in the bar at the bottom and selecting “Configure Platforms”. In the new window, set the FPGA bitfile location, save the changes and program the FPGA.

Finally, execute the program. For that purpose, click on the “Run and Debug” button, go to the LAUNCH-SELECT section and select the .elf file which you just built (or select “follow cmake target”). Start the debugger by clicking on the play button ▷ (Embedded) Attach.

Once the program starts executing, you can do as many tests as you wish: run step-by-step, inspect memory, the peripherals and the registers, view the disassembly code, etc.

Step 4. Compile, download, and run a C program on RVfpgaEL2-ViDBo

You can also test execution on RVfpgaEL2-ViDBo. For that purpose, copy the **commandLine** folder provided at *[RVfpgaBooleanPath]/Labs/Lab01/ProjectSources* into your project folder (i.e. *[RVfpgaBooleanPath]/Labs/Lab01/Project1*), and then compile the program and launch it on RVfpga-ViDBo as explained in detail in Section 7 of the GSG. Confirm that when a switch changes its state, the corresponding LED also changes its state.

3. Exercises

Create your own C programs by completing the following exercises. Note that if you leave the Boolean board connected to your computer and powered on, you do not need to reload RVfpgaEL2-Boolean onto the board between running different programs. However, if you turn off the Boolean board, you will need to reload RVfpgaEL2-Boolean onto the board using Catapult.

Remember that you can print any variable using the PSP/BSP functions. Remember as well that you can run these programs in simulation, using the different Verilator-based simulators and the RVfpgaEL2-Whisper Instruction Set Simulator.

Exercise 1. Write a C program that flashes the value of the switches onto the LEDs. The value should pulse on and off slow enough that a person can view the flashing. Name the program **FlashSwitchesToLEDs.c**.

Exercise 2. Write a C program that displays the inverse value of the switches on the LEDs. For example, if the switches are (in binary): 0101010101010101, then the LEDs should display: 1010101010101010; if the switches are: 1111000011110000, then the LEDs should display: 0000111100001111; and so on. Name the program **DisplayInverse.c**.

Exercise 3. Write a C program that scrolls increasing numbers of lit LEDs back and forth until all of the LEDs are lit. Then the pattern should repeat. Name the program **ScrollLEDs.c**.

The program should cause the following to occur:

1. First, one lit LED should scroll from right to left and then left to right.
2. Then two lit LEDs should scroll from right to left and then left to right.
3. Then three lit LEDs should scroll from right to left and then left to right.
4. And so on, until all the LEDs are lit.
5. Then the pattern should repeat.

Exercise 4. Write a C program that displays the unsigned 4-bit addition of the 4 least significant bits of the switches and the 4 most significant bits of the switches. Display the result on the 4 least significant (right-most) bits of the LEDs. Name the program **4bitAdd.c**. The fifth bit of the LEDs should light up when unsigned overflow occurs (that is when the carry out is 1).

Exercise 5. Write a C program that finds the *greatest common divisor* of two numbers, *a* and *b*, according to the Euclidean algorithm. The values *a* and *b* should be statically defined variables in the program. Name the program **GCD.c**. Here is some additional information about the Euclidean algorithm: https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm. You can also simply google "Euclidean algorithm".

Exercise 6. Write a C program that computes the first 12 numbers in the Fibonacci sequence and stores the result in a finite vector (i.e. array), *V*, of length 12. This infinite sequence of Fibonacci numbers is defined as:

`V(0)=0, V(1)=1, V(i)=V(i-1)+V(i-2)` (where `i=0,1,2...`)

In words, the Fibonacci number corresponding to element i is the sum of the two previous Fibonacci numbers in the series. Table 1 shows the Fibonacci numbers for i = 0 to 8.

Table 1. Fibonacci series

i	0	1	2	3	4	5	6	7	8
V	0	1	1	2	3	5	8	13	21

The dimension of the vector, *N*, must be defined in the program as a constant. Name the program **Fibonacci.c**.

Exercise 7. Given an *N*-element vector (i.e., array), *A*, generate another vector, *B*, such that *B* only contains those elements of *A* that are even numbers greater than 0. The C program must also count the number of elements in *B* and print that value at the end of the program. For example: suppose *N*=12 and *A* = [0,1,2,7,-8,4,5,12,11,-2,6,3], then B would be: *B* = [2,4,12,6]. Because B has four elements, the following should print at the end of the program:

```
Number of elements in B = 4.
```

Use the `ee_printf` function to do so. Name the program **EvenPositiveNumbers.c**. Test your program when *A* has 12 elements.
Exercise 8. Given two *N*-element vectors (i.e., arrays), *A* and *B*, create another vector, *C*, defined as:

```
C(i) = |A[i] + B[N-i-1]|, i = 0,..,N-1.
```

Write a program in C that computes the new vector. Use 12-element arrays in your program. Name the program **AddVectors.c**.

Exercise 9. Implement the bubble sort algorithm in C. This algorithm sorts the components of a vector in ascending order by means of the following procedure:

1. Traverse the vector repeatedly until done.
2. Interchanging any pair of adjacent components if `V(i) > V(i+1).`
3. The algorithm stops when every pair of consecutive components is in order.

Use 12-element arrays to test your program. Name the program **BubbleSort.c**.

Exercise 10. Write a program in C that computes the factorial of a given non-negative number, *n*, by means of iterative multiplications. While you should test your program for multiple values of n, your final submission should be for *n* = 7. The program should print out the value of factorial(*n*) at the end of the program. *n* should be a variable that is statically defined within the program. Name the program **Factorial.c**.