System programming

Exercises

Implement the following tasks under Linux operating system using C programming language.

1. Write a program, which prints all its command-line arguments which are started by hyphen (`'-'`) or slash (`'/'`) characters (these are usually called switches of the program).
2. Write a program, which can assume that is command-line argument is formally a positive integer number, and the program prints out the square of this value.
3. Write a program, which makes a copy of its own executable file. The copy must be called `"last"`. (Do not forget, that the name of the executable is not known at the moment of program development.)
4. Write a program, which returns to the operating system by the number of characters (excluding white spaces) within the command launching the program itself. (For example returns by 12 in case of the `"./a.out Hello"` command.)
5. The `gcc` command has many potential command-line arguments, but let's focus on only the following three (which have arbitrary sequence/order.
   - The `gcc` needs a source filename. This is a string, which cannot be started by a hyphen (`'-'`) character. (This will be complied to object code and then to executable.) If the source filename is absent, we get an error message.
   - The `"-Wall"` switch causes fully printed warning messages of the compilation It is an optional switch.
   - The `"-o"` switch is followed by a string which will be the name of the executable file (not the source file) after the compilation. If this switch is missing then the name of the executable is always `"a.out"`. If the switch is present but it is not followed by a filename string, then we got an error message.

   Write a program, which can manage the above situations, but it does not make a real compilation just prints the meaning of the current command. For example, some execution commands (including but not limited to all the possibilities) must be handled by the program according to the above description (supposing that the name of the executable is `"myprog"`):

   ```
   ./myprog                      ./myprog -o run ab.c

   ./myprog ab.c                 ./myprog -o run ab.c -Wall

   ./myprog -Wall ab.c           ./myprog -Wall -o run ab.c

   ./myprog ab.c -o run          ./myprog -o ab.
   ```
6. Write a program, which is able to tell whether we are in the default directory of the current user. (The `HOME` and the `PWD` environment variables can help.)
7. Write a program, which prints a text following the `"<user>@<host>"` format, where the `"<user>"` is the name of the current user, and the `"<host>"` is the name of the presently used computer.
8. Write a program, which displays all the available environment variables on the standard error interface.
9. Write a program, which is a simple „Hello World" program in the case of the Linux operating system but does not print anything to the screen in the case of other operating systems.
10. Write a program, which returns by the number of seconds remaining from the current minute according to the system time.

11. Write a program, which prints that the program is present execution happens on a weekday or at weekend.
12. Write a program, which asks the user to type a long word then the program must tell how many seconds it took to type (with ± 1 second tolerance)!
13. Write a program, which generates two real random numbers in the [0.0 ; 1000.0] closed interval, it prints a third random number which is between the values of the previous two.
14. Write a program, which prints a maximum 6 digit long positive integer number. When the program is executed again in the same minute it must print the same number, but it prints different numbers in each minute.
15. Write a program, which simulates a tricky roulette game, that is it prints randomly an integer between 0 and 36 (including them as well), but the number 17 has 3% probability while the other values have a lower, equal chance.
16. Write a program, which prints 1000 characters to the screen. These are the lower-case English letters or the space character. The probability of the letters is seven times larger than the chance of the space.
17. The two-dimensional random walk is a process, where we take a 1-unit long step in each time moment in the 4 possible directions (up, down, left, right) randomly. Write a program, which simulates such a random walk. The steps must be repeated until our planar distance from the initial position reaches the 100 units. Finally, the program prints out the number of necessary iterations.
18. We are playing a 'head-or-tail' game. In each round, we bet for the head using 1 coin as a stake. If we get head, we will gain one coin, contrary we will lose one. Initially, we have 10 coins. We play until either we lost all our coins, or we double our money. Write a program to play. Finally, the program prints out the number of played rounds.
19. Write a program, which prints 1000 digits (0-9) to the screen randomly, but not with uniform distribution. The probability of each digit must be proportional to its own values. For example, the 4 appears with a double chance compared to the 2, and the frequency of the 3 is one-third of the frequency of the 9.
20. Write a program that simulates a Scandinavian lottery draw, i.e., writes out 7 integers between 1 and 35 (including them and considering that repetition is not possible)!
21. Write a program, which contains the definition of a function related to string operations. Its only one parameter is the address of a string, and its return value is a pointer to refer to the address of the first non-letter character in the string. (Letters mean the upper-case and lower-case characters of the English alphabet.) When the string contains only letter characters the function returns with `NULL` pointer value. In the main program unit, read a string from the keyboard and replace all the non-letter characters of this string to a space character using your function.
22. Write a program which contains the definition of a procedure having 2 parameters. The procedure swaps the values of the parameters. The parameters are addresses of two integer variables, where the values to swap locate. In the main program unit, sort an integer array using this procedure
23. Write a program which contains the definition of a procedure having 2 parameters. The procedure sorts the elements of an array storing a real number. (You can choose any sorting algorithm.) The first parameter of the procedure is the address of a `float` type array, the second parameter is the number of real numbers in the array. In the main program unit, create an array with arbitrary content and sort it with your procedure, then print the sorted values.
24. Write a program, which contains the definition of a function. It has 4 parameters, the starting addresses of two integer arrays and the numbers of values stored in these arrays. (The size of the arrays can be different.) The function must allocate a continuous memory space for all the elements of both arrays. The subroutine copies all the values (in arbitrary order) to this memory space and then it returns by the starting address of this RAM space. In the main program unit create 2 different arrays,

call the procedure, and then print the values from the memory address provided by the function. All dynamic memory allocation must be free before the program terminates.

25. Write a program, which uses a queue according to the First In First Out operation based on a linked-list. In the program use an `int` type variable as a counter starting from 1. A current value of this variable must be put to the end of the queue and then the variable is incremented. During the execution of the program first put the value of the variable 10 times into the queue (10 consecutive integers) then start a loop. In all iteration cycle, eighter get the first value of the queue and print it, or put the current value of the counter into the queue. Both cases must be 50%-50% probability randomly. The iteration terminates if eighter the queue becomes empty of you reach the 1000 in the counter variable. The memory which is allocated during the execution finally must be free.
26. Write a program, which stores integer values in a one-directionally linked-list. The values must be stored/inserted in increasing order along the direction of the linkage. Store 100 random values, then go through the list and print the values separated by a space. The memory which is allocated during the execution finally must be free.
27. Write a program, which uses a two-directional linked-list to store characters. (Naturally, the list needs a *head* and an *end* pointer.) Store the letters of your name in the list, then print the content of the list, first in the head-to-end direction, then in the opposite end-to-head direction. Finally, the memory allocated by the program, must be free.
28. Write a program, which registers in a `char` type variable (using bit operations) that an employee works or not on the days of a week. The least significant bit of the byte can mean the Monday and the second most significant bit means the Sunday together with if the bit is `1` in a given position it means the employee works on that day, if it is `0`, he/she does not work that day! In the program create two `int` type functions that have only one `char` type parameter as a week register. One of the functions must return the number of weekdays when the employee works, the other function returns the number of weekend days when he/she does not work. In the main program unit initialize a character variable by a random number between 0 and 127, call the functions for this value and print the values provided by the functions.
29. An uncompressed image with 2-bit color depth stores the index of the 4 possible colors of each pixel on 2 bits (binary: 00, 01, 10, 11) continuously in row order. (It is called pixel array.) Write a program, in which first you create an imaginary pixel array, that is you allocate a 100 bytes long memory field (array), then you initialize the elements by random bytes between 0 and 255. After this, the program must print the number of occurrences of all the 4 colors. (For example, in a 2 bytes long array, when the values of these two bytes are 79 and 203, that is in binary 01001111 11001011, then the frequency of each color index is the following 00: 2 pixels, 01: 1 pixel, 10: 1 pixel, 11: 4 pixels.)
30. In the case of the packed BCD representation a decimal digit is stored on 4 bits (according to the binary form of the digit). In this way, 2 digits are stored separately in a byte (if needed with leading zeros). Write a program, in which you create a procedure having an unsigned integer parameter. The procedure prints out this integer value as a packed BCD coded character sequence using the required number of bytes. Call the procedure in the main program with an arbitrary positive integer number. Some examples:

parameter	*BCD coded bit sequence*	*character codes*	*characters*
`41`	`01000001`	`65`	`"A"`
`2340`	`00100011 01000000`	`35, 64`	`"#@"`
`672538`	`01100111 00100101 00111000`	`103, 37, 56`	`"g%8"`

31. During the UTF-8 coding a 2-byte Unicode of a character (`unsigned short int`) is converted to a byte sequence (considerable as a character sequence), which is 1, 2, or 3 bytes long. The conversion process depends on the magnitude of the Unicode value, it inserts special bits to given positions according to the following patterns (https://en.wikipedia.org/wiki/UTF-8):

unicode domain	*binary unicode form*	*binary UTF-8 form*
`0-127`	`00000000 0abcdefg`	`0abcdefg`

128-2047	00000abc defghijk	110abcde 10fghijk
2048-65535	abcdefgh ijklmnop	1110abcd 10efghij 10klmnop

Write a program, in which there is a procedure. It prints its unsigned integer parameter (unicode value) using bit operations as a character sequence according to the UTF-8 coding applying the necessary number of characters. Call the procedure in the main program unit with a random positive integer not greater than 65535. Some examples:

parameter	*unicode bits*	*UTF-8 coded bits*	`char` *values*
66	00000000 01000010	01000010	66
734	00000010 11011110	11001011 10011110	203, 158
22904	01011001 01111000	11100101 10100101 10111000	229, 165, 184

32. Write a program, in which there is a function using bit operations to change the byte order of the only one `int` type parameter. (The order of bits inside each byte does not change.) The function must return the reversed byte order value. In the main program unit, call the function with an arbitrary parameter and pass the return value as a parameter for the second call of the function. Print out the initial value, and the return values of the first and second calls. For example: 308 276 097 → 2 179 686 162 → 308 276 097.

33. The IPv4 network addresses are unsigned fixed-point represented values on 32 bits (`unsigned int`), which is used in dotted-decimal form for human users. This means that each byte is written separately in decimal form and the 4 integers are separated by dots. Write a program using bit operations and it contains a procedure and a function. The procedure has an unsigned int parameter which is printed to the screen in dotted decimal form. The function has 4 integer parameters, and it transforms these values to an `unsigned int` return value supposing that the parameters are the small integers in a dotted-decimal IPv4 address. In the main program unit, call the function passing 4 values between 0-255 reading from the keyboard. Then the return value must be passed as a parameter to the procedure.

34. The netmask which is applied in the case of IPv4 network addresses is a 32 bits long value that has two bit-regions. At the beginning of the mask, there are only 1 bits while at the end only 0 bits are present. The prefix length of the mask is the number of 1 bits in the netmask. Write a program that reads an integer number between 1 and 31 from the user and prints that unsigned integer value in a decimal form which can be a netmask with the given prefix length. For example: 21 → 11111111 11111111 11111000 00000000 → 4294965248

35. There are integer values in a text file separated by space characters. Their number is unknown. (The name of the file is given by you.) Write a program which saves some of the numbers stored in the file (keeping the old content as well), to the end of the file anytime the program is executed.

36. There are 3 digits long integer numbers in a text file separated by space characters. Write a program which goes through all the numbers in the file and if any of them is started by a '1' digit it overwrites the number with its double. All the other values remain the same.

37. Write a program that stores 100 random integers greater than minus one million into an array of integers. The program must contain 2 subroutines. A procedure, which gets the previous array as a parameter and prints these values to a text file using formatted I/O. A function, without parameters returning the 79th number from the file. In the main program unit, call the procedure to generate the values and call the function and print the return value to the screen.

38. Write a program that stores 100 random integers greater than minus one million into an array of integers. The program must contain 2 subroutines. A procedure, which gets the previous array as a parameter and prints these values to a binary file using low-level I/O. A function, without parameters returning the 79th number from the file. In the main program unit, call the procedure to generate the values and call the function and print the return value to the screen.

39. From the 43rd byte of a BMP image file the next 8 bytes stores the width and the height of the image as 2 little-endian unsigned fixed-point values on 32 bits. Write a program that gets the name of a BMP image as a command-line argument and prints the resolution of the image.

40. A traditional WAV sound file stores the sample rate of the sound in Hertz from the 25th byte to the 28th byte of the file, on 32 bits as a binary unsigned, fixed-point integer in little-endian byte order. Write a program that gets the name of a WAV file as a command-line argument, and it prints the sample rate of the sound to the screen.
41. Write a program, in which we would like to use coherent values in a record (`struct`). The record must have an `int`, a `float`, a `double`, a `char`, and a 10-items long char array fields. You must initialize them by arbitrary values. After this, the program must save the total content of the record into a binary file using low-level file handling. Write another program that also knows the above-defined record and reads the binary file content to a `struct` variable and prints the fields to the screen.
42. Save a BMP image from the internet with at least 100×100 pixel resolution. Write a program, which overwrites the last 1000 bytes of this binary file with 0x55 hexadecimal values for each byte. Look at the modified image.
43. Write a program, which prints the files of your current working directory having `".txt"` extension. The filenames must be separated by a space.
44. Write a program, which contains a function returning with an integer number and having a parameter the starting address of a string. The function returns by 1, if the parameter points to a string which is a file or directory name within the default directory of the current user, otherwise, the function returns by 0. In the main program unit, ask a string from the user and call the function passing it as a parameter. Depending on the return value print the "Exists" or the "Not exists" text.
45. Write a program which prints 2 numbers to the output. One of them is the number of files in the current directory and the other is the number of subdirectories.
46. Write a program, which determines the total size of the files in the parent directory of the current location and print it in kilobyte unit to the screen.
47. Write a program, which prints to the standard output the names of those files within the "Desktop" folder of the current user which are modified in the last hour.
48. Write a program, which prints the names of those files of the current folder that have execution rights for all users. (Do not display the name of subdirectories.)
49. Write two programs, which can communicate via the Internet using sockets. When the server receives a UDP segment from the client which contains a "Who are you?" string then it sends back the username of the current user.
50. Write two programs, which can implement a rock-paper-scissors game by socket programming. One of the programs is a TCP server which receives the connection requests of the two client processes and then sends a "start of game" message to the clients. Users of the clients (players) enter a hand sign (rock, paper, or scissors) and the clients send them to the server. After the server receives the two messages from the client it sends a response to the players to tell who the winner is and then they close the connections.
51. Write a program, which creates a child process with `fork` operation, and then the child also creates an own child. The "grandchild" process must print out the process ID (PID) of its "grandparent" process.
52. Write a program, which creates two child processes with `fork` operations. The "younger child" process must print out the process ID (PID) of its "older brother's" process.
53. Write a program, which creates a child process with `fork` operation then both processes must print out the sum of integers from 1 to their own PID.
54. Write a program, which creates a pipe, and then it creates a child process by a `fork` operation. The child process must send its own PID to the parent via the pipe. The parent must compare the value received by the pipe and the return value of the fork call and displays the result.

55. Write a program, which creates a child process by `fork` operation. The parent must ask for a number from the user and it saves it to a text file called ”`number.txt`”. The child reads the number from the file and writes it to the standard output. You must synchronize the processes by “user 1” signal for successful communication.

56. Write a program, which makes up 2 positive integers with a maximum of 3-digit positions printing them to the screen. The program must ask the sum of the numbers from the standard input (the user). If the user enters the sum within 3 seconds, the program must check its correctness, otherwise, the program prints out a “Slow” word and terminates.

57. Write a program, which increments an integer variable in an infinite loop as fast as possible. When the user push a `ctrl+c` keyboard combination, the program prints the current value of the variable before it terminates.

58. Write a procedure in your program, which behaves in a similar way to the built-in `sleep()` library function. That is, it waits for the time specified as a parameter and after this amount of time (seconds) the procedure terminates without doing further activities. (Use signal and timer.) Call the procedure in the main program unit.

59. Forge complex real-life problems where their solutions require tools to be used during the semester. Implement the solutions.