2-week HOME EXAM
PG3401 C Programming
Permitted aids: All
Duration: 14 days
Grading scale/evaluation type: National grading scale A - F
Date: April 30th - May 14th, 2025

The exam text is in English only, you can hand in your exam in either English or Norwegian (Swedish and Danish is also accepted). Programming language used should though be the language C-89, other languages than C will not earn points.

The exam text has 10 pages. There are a total of 6 tasks in the exam set.

There is a 2-week deadline for this home exam. The period may overlap with other exams you have, so it is important that you use the time efficiently and plan your time well, so that you do not sit right before the deadline and have to submit several exams at the same time. Please note that the exam MUST be submitted within the deadline set in Wiseflow, and the assignment can only be submitted via WISEFLOW. It will not be possible to submit the assignment after the deadline – this means that you should submit well in advance so that you can contact the exam office or support if you have technical problems.

It is emphasized that the student must answer the exam independently and individually; cooperation between students and plagiarism is not allowed, the answer should represent the student's own work. This also means that it is not allowed to use code- or text-generating tools based on artificial intelligence. Be aware that attempts at "bluffing" on practical assignments in the form of screenshots found on the internet, screenshots received by other students or edited screenshots give negative points on the total score on the exam (which may result in the final grade being reduced by one or more grades), and in serious cases it will result in the filing of a cheating/plagiarism case. You should only document what you have achieved yourself, not try to make it look like you have achieved something you have not actually achieved. The exam must be solved on Linux.

Note that the tasks are made with increasing difficulty, and especially the last three tasks are more difficult than the first tasks. It is therefore encouraged to finish the first assignments (completely) so that the student does not spend all the time doing the most difficult tasks first.

Format of submission

This is a practical programming exam (apart from task 1), so the focus should be on explaining how you have proceeded to solve the tasks, justifying your choices and presenting any assumptions you have made in your solution.

If you are unable to solve a task, it is better if you submit what you have done (even if it does not work), and explain how you have tried to solve problems you faced and what you did not achieve – than not answer the task at all. It is expected that everything works unless otherwise described in the PDF file, code that the student knows does not work should also be commented as such in the code. If you know that the program is crashing, not compiling or not working as intended, it is important to explain this along with what steps you have taken to try to solve the problem.

The submitted source code must be in 1 ZIP file, the name of the file must be PG3401_V25_[candidate number].zip. This file should have the following structure:
\ logdata \ ewa_log.txt
\ logdata \ ewa_data.bin
\ task2_[candidate number] \ makefile
\ task2_[candidate number] \ […]
\ task3_[candidate number] \ makefile
\ task3_[candidate number] \ […]
[…]

In Wiseflow, you should upload a text answer with the name "PG3401_V25_[candidate number].pdf", and the ZIP file should be uploaded as an attachment to the text answer.

Be sure all files are included in the ZIP file (this includes all executable files, all makefiles, as well as the contents of the logdata folder). Each task-folder should have a makefile file, and no changes, third-party components or parameters should be required - the assessor will in shell on Debian Linux 10 go into the folder and type "make" and this should build the program with GCC. On all programming tasks, it is an absolute requirement to use the makefile file used during this course – as showed on Lecture 4, slide 65 with the title «Use this makefile», you can make minor changes to the makefile file if necessary to solve the task, however, tasks that use a different makefile, regardless of whether it is auto-generated or created manually, will result in no points being awarded on the task. The referred lecture is available at the following URL: http://www.eastwill.no/pg3401/pg3401_lecture04.pdf

The text answer should contain an answer to assignment 1 (write it to the point, does not need a long thesis – however it should still be an answer of academic quality and style as expected of all work at college level). After the plain text answer, there should be 1 page of justification / documentation for each assignment, each of the documentation segments should be on a new page to make the text answer clear to the assessor. The answer must be in PDF format and have the correct file extension to be opened on both a Linux and a Windows machine (.pdf). Answers in other formats will not be read. (The text answer in the PDF file can be written on Mac or Windows, or any other operating system the student may be using, installing word processor on Linux is not required for this course.)

Required preparation

Before you start solving the programming tasks you need to download the tool EWA; this is new for this year's exam and has been covered during LECTURE 9 (streamed from Oslo March 20th) and demoed in the workshop held in Bergen April 3rd.

You can either download the application from the domain 'eastwill.no' or download it from Canvas where it is found on the PG3401 coursepage under EKSAMEN (EXAM).

The student MUST use Debian 10 64 bit compiled for Intel (for students using VmWare Workstation on Windows or other Intel based systems) or compiled for ARM (for students using VmWare Fusion on Mac using M1, M2, M3 or M4 systems). No other versions of the application will be provided during the exam as only these 2 systems are supported.

To download from 'eastwill.no' follow these steps, the file must be downloaded, then unpacked, and execute privileges must be set manually. Note that it MUST be run from a location on your system where you have write-access or it will fail (no help will be given during the exam, understanding how to do this is required knowledge).

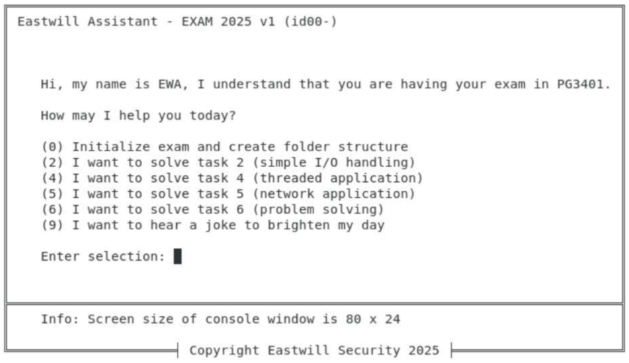
ZIP file is also found under EKSAMEN on Canvas, it can be copied in manually (from a USB drive for instance) if you don't have internet access in your VM. Access to the internet is not required for any task – but a functioning network adapter that can communicate on localhost (127.0.0.1) is required for Task 5 (the network task).

VmWare Workstation (Intel based) or x64 Fusion (Mac with Intel CPU):

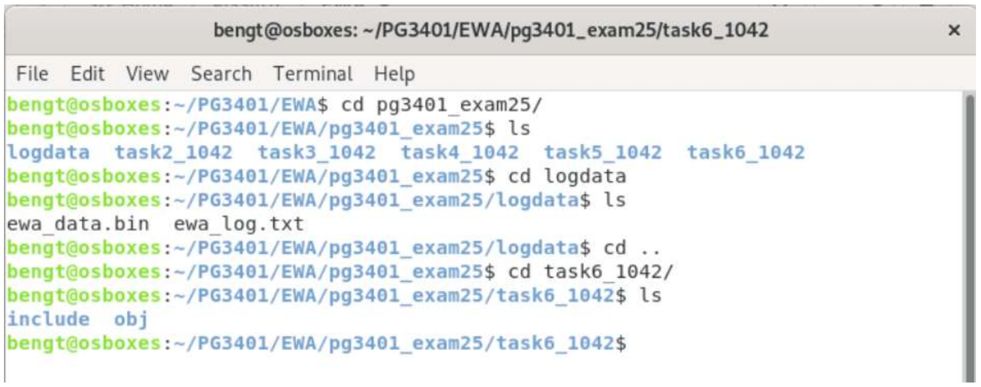
```
wget http://www.eastwill.no/pg3401/exam25/ewaexamv1_exam25_intel64.zip
unzip ewaexamv1_exam25_intel64.zip
chmod +x ewa_exam_v25_i64
./ewa_exam_v25_i64
```

VmWare Fusion (ARM based):

```
wget http://www.eastwill.no/pg3401/exam25/ewaexamv1_exam25_arm64.zip
unzip ewaexamv1_exam25_arm64.zip
chmod +x ewa_exam_v25_a64
./ewa_exam_v25_a64
```


```
Eastwill Assistant - EXAM 2025 v1 (id00-)


   Hi, my name is EWA, I understand that you are having your exam in PG3401.

   How may I help you today?

   (0) Initialize exam and create folder structure
   (2) I want to solve task 2 (simple I/O handling)
   (4) I want to solve task 4 (threaded application)
   (5) I want to solve task 5 (network application)
   (6) I want to solve task 6 (problem solving)
   (9) I want to hear a joke to brighten my day

   Enter selection: █

   Info: Screen size of console window is 80 x 24
                      ┤ Copyright Eastwill Security 2025 ├
```

Select option 0 to initialize the exam, when prompted enter your candidate number on this exam. This will create the folder structure you will use on this exam which should look like this:



```
bengt@osboxes: ~/PG3401/EWA/pg3401_exam25/task6_1042
File  Edit  View  Search  Terminal  Help
bengt@osboxes:~/PG3401/EWA$ cd pg3401_exam25/
bengt@osboxes:~/PG3401/EWA/pg3401_exam25$ ls
logdata  task2_1042  task3_1042  task4_1042  task5_1042  task6_1042
bengt@osboxes:~/PG3401/EWA/pg3401_exam25$ cd logdata
bengt@osboxes:~/PG3401/EWA/pg3401_exam25/logdata$ ls
ewa_data.bin  ewa_log.txt
bengt@osboxes:~/PG3401/EWA/pg3401_exam25/logdata$ cd ..
bengt@osboxes:~/PG3401/EWA/pg3401_exam25$ cd task6_1042/
bengt@osboxes:~/PG3401/EWA/pg3401_exam25/task6_1042$ ls
include  obj
bengt@osboxes:~/PG3401/EWA/pg3401_exam25/task6_1042$
```

Task 1. Theory (5 %)

a) Explain what the C programming language can be used for.

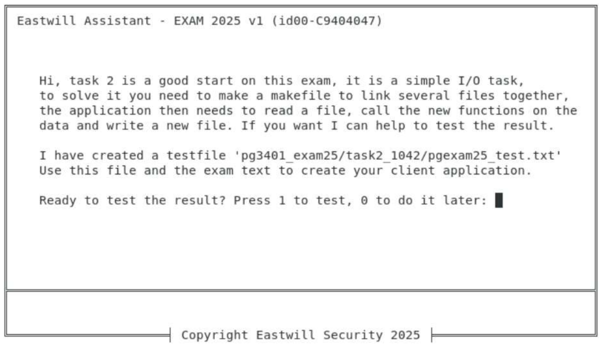
b) Who is Dennis Ritchie and what is he known for in the field of Information Technology?

c) List at least 5 different Linux "distros", explain briefly their background, if they have any specific purpose, and how they differ from each other.

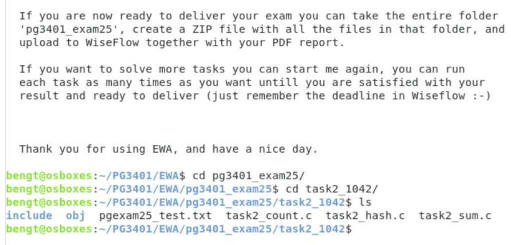
Task 2. File management and functions (15 %)

For this task you must first do the "Required preparation" as described above. In the EWA tool make sure you have chosen 0 to create and initialize the folder structure, and have supplied your candidate number for this exam.

In the EWA main interface select 2. This will create 3 source files that you will use for this task as well as a test file. The resulting output should look something like this:



```
Eastwill Assistant - EXAM 2025 v1 (id00-C9404047)


   Hi, task 2 is a good start on this exam, it is a simple I/O task,
   to solve it you need to make a makefile to link several files together,
   the application then needs to read a file, call the new functions on the
   data and write a new file. If you want I can help to test the result.

   I have created a testfile 'pg3401_exam25/task2_1042/pgexam25_test.txt'
   Use this file and the exam text to create your client application.

   Ready to test the result? Press 1 to test, 0 to do it later: █

                    ┤ Copyright Eastwill Security 2025 ├
```



```
  If you are now ready to deliver your exam you can take the entire folder
  'pg3401_exam25', create a ZIP file with all the files in that folder, and
  upload to WiseFlow together with your PDF report.

  If you want to solve more tasks you can start me again, you can run
  each task as many times as you want untill you are satisfied with your
  result and ready to deliver (just remember the deadline in Wiseflow :-)



  Thank you for using EWA, and have a nice day.

bengt@osboxes:~/PG3401/EWA$ cd pg3401_exam25/
bengt@osboxes:~/PG3401/EWA/pg3401_exam25$ cd task2_1042/
bengt@osboxes:~/PG3401/EWA/pg3401_exam25/task2_1042$ ls
include  obj  pgexam25_test.txt  task2_count.c  task2_hash.c  task2_sum.c
bengt@osboxes:~/PG3401/EWA/pg3401_exam25/task2_1042$
```

/pg3401_exam25/task2_[candidate number]/task2_hash.c
/pg3401_exam25/task2_[candidate number]/task2_count.c
/pg3401_exam25/task2_[candidate number]/task2_sum.c
/pg3401_exam25/task2_[candidate number]/pgexam25_test.txt

You should write a program that reads the .txt file and calls the above functions in the 3 created source files to calculate metadata about the file: a) a DJB2 hash, b) the count of each letter, c) size in bytes and a sum of all characters. You also need to create 1 header file for each of these source files and include these header files from your main source file.

The result of these function calls should be stored in the following struct:

```
struct TASK2_FILE_METADATA {
   char szFileName[32];
   int iFileSize;
   char byHash[4];
   int iSumOfChars;
   char aAlphaCount[26];
}
```

The program should create an output file where this structure is written as BINARY, this file must be named pgexam25_output.bin and be placed in the same folder as the input file the EWA tool has created for you.

Required documentation:

You must make a screenshot of YOUR program when it is running, you must also make a screenshot of EWA after you have pressed 2 and it has created the files you will use in your program. You also need to run the EWA tool again after you have completed your program and then ask EWA to verify your output file by pressing 1 – and make a screenshot after EWA has verified it. The screenshots need to be in the ZIP file and be named task2_scrn1.png, task2_scrn2.png and task2_scrn3.png (and should be placed in the same folder as your makefile) AND it must also be included in a proper location in your PDF file.

The output file pgexam25_output.bin must be part of the submitted ZIP file.

Task 3. List handling (20 %)

You will be creating a simple data structure to handle flight-departures. The "core" of the list should be a double-linked list of planes, each item (struct) in the list should contain pointers to both the previous item and the next item. The item should also contain a text string that is FLIGHTID (for example, BA-42), a text string describing DESTINATION, an integer for the number of SEATS, and an integer that should serve as the TIME for departure. Each item should additionally contain a single-linked list of (reservations for) PASSENGERS.

Each item in the passenger list (reservations) should contain an integer containing the SEAT NUMBER, a text string with the NAME, and an integer containing the AGE of the traveler. The list should always be SORTED by seat number.

You will create functions that perform the following operations on the list:

- The code must include comments that explain what the code does, and be structured and organized to improve readability and maintainability
- Add a flight to the list
- Add a passenger to a flight-departure (remember that the list should always be sorted by seat number)
- Retrieves a flight N from the list (counted from the start of the list, where the first item is item number 1) and prints on the screen all the data associated with the departure including the list of passengers
- Finds flight that matches departure DESTINATION in the list, and returns what "item number" it has – re function above
- Deletes a flight (and all passenger reservations for the departure)
- Changing the seat of a passenger
- Search through the lists for a passenger's NAME (in all flights) and print on the screen all flights this passenger is associated with
- Search through the lists any passenger (in any flights) that are booked on more than 1 flight, and print the name of the passenger on the screen

You are going to create a main function that receives instructions from the user based on input from the keyboard, the main must be able to call all eight functions above (for example, in some form of menu) and ask the user for data necessary to call the mentioned functions. Main should clean up all data before it returns (an option in the menu must be to exit).

Required documentation:

You must make one screenshot when your application starts (showing the menu screen), and one of the application when it is running one of the “tasks”. The screenshot needs to be in the ZIP file and be named task3_scrn1.png and task3_scrn2.png (and should be placed in the same folder as your makefile) AND it must also be included in a propper location in your PDF file.

Task 4. Threads (20 %)

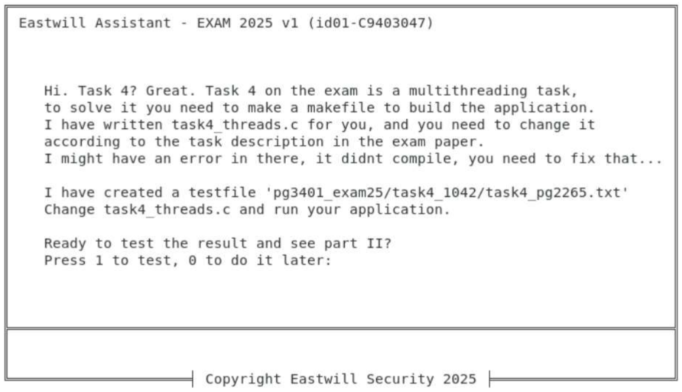
In practical programming, it is often efficient to put time-consuming operations in worker-threads, examples of which are file operations, network operations and communication with external devices. In this assignment, you will simulate such operations with a smaller data set – in order to save you time during the exam, a ready-made application has already been created for you to change.

Application consists of 3 threads, the main thread (running the main function) and 2 worker threads. The main thread starts the worker threads with mechanisms for thread communication and synchronization (in this application, the main thread has no other function, and only starts the two threads that do the job themselves). The threads have a memory area with space of 4096 bytes for communication between the two threads.

One worker thread (thread A) read a text file, worker thread A should then send the file over to the other worker thread (thread B) through multiple cycles using the memory space described above, signaling thread B that there is data available in the buffer. Thread B then counts the number of instances of bytes with value 00, 01, 02, including; FF in the file it receives from thread A. Thread A and thread B both loop to process the file until it's finished. When the file is sent over in its entirety, *worker thread A will end. Worker thread B completes its count of bytes, and then prints to the terminal window the number of instances of each of the 256 possible byte values, before it also exits. The main thread waits for both threads to exit, cleans up correctly, and then closes the application.*

For this task you must first do the “Required preparation” as described above. In the EWA make sure you have chosen 0 to create and initialize the folder structure, and have supplied your candidate number for this exam.

In the EWA main interface select 4. This will create 2 files that you will use for this task; a source file with the code described above, and a txt file to be used for testing.



```
Eastwill Assistant - EXAM 2025 v1 (id01-C9403047)


   Hi. Task 4? Great. Task 4 on the exam is a multithreading task,
   to solve it you need to make a makefile to build the application.
   I have written task4_threads.c for you, and you need to change it
   according to the task description in the exam paper.
   I might have an error in there, it didnt compile, you need to fix that...

   I have created a testfile 'pg3401_exam25/task4_1042/task4_pg2265.txt'
   Change task4_threads.c and run your application.

   Ready to test the result and see part II?
   Press 1 to test, 0 to do it later:

                    Copyright Eastwill Security 2025
```

/pg3401_exam25/task4_[candidate number]/task4_threads.c
/pg3401_exam25/task4_[candidate number]/task4_pg2265.txt

The source file task4_sample.c is a solution to the application as was described above. The test file task4_pg2265.txt is the file you will use to test your multithreaded application (Hamlet by Shakespeare, taken from Project Gutenberg – originated from https://www.gutenberg.org/ebooks/2265).

PART I

The following changes must be done in task4_sample.c, do NOT create new files, and do NOT move the file to other folders (the EWA tool will read the file when you are ready, and expects it to be at the same location/file as it created it):

- You must create a makefile file (as detailed above under “Format of submission”) to build the program, make any changes required to the makefile file and source file to make the program build and run correctly
- The program uses global variables, change this to that all variables are local variables in the main function and submitted as parameter to the two threads
- Thread A has hardcoded the name of the file to be read, change the program so it takes the file name as a parameter on the command line and passes this on to Thread A as a variable
- Rewrite the code from using conditions to using semaphores
- Add code for explicit initialization of mutex and semaphores (with the *_init functions)
- Add comments to your code to document what the code does

PART II

- If you are able to complete the first part of the task, the tool EWA will prompt you for two (2) more changes you will make to your final solution.
- You need to (at least) partially succeed on Part I to be shown Part II in the EWA tool, this is required because Part II builds upon Part I of this task
- Once you have succeeded sufficiently the EWA tool will give you the next part of this task – your progress is shown in the tool

Required documentation:

You must take a screenshot of the EWA tool after you have completed part I (if you were able to do so). Your final ZIP file must include all files, including any output files from Part II (if you were able to complete Part II).

Task 5. Network (20 %)

For this task you must first do the “Required preparation” as described above. In the EWA make sure you have chosen 0 to create and initialize the folder structure, and have supplied your candidate number for this exam.

In the EWA main interface select 5. This will create a header file and a sample file:

```
/pg3401_exam25/task5_[candidate number]/ pgexam25_task5.h
/pg3401_exam25/task5_[candidate number]/ pgexam25_sample.txt
```

In this header file you will find some define values and data structures. You are going to create a server application (and the EWA tool has implemented a client that will be used for testing your server). In order to run this you need to open 2 terminal windows; one running your application, and one running EWA.

The server application should be executed with port number and a user-specified ID on the command line, such as "task5 -port 25 -id SmtpTest". When started, the application should open the specified port for LISTEN, for this task it is enough to bind to loopback on 127.0.0.1, and communication must use TCP.

You should simulate traffic between client and server (the traffic is “inspired by SMTP”), points will be given for implementations that are flexible and robust, with error handling that can correctly handle data received from the client (especially if it is wrong – or malicious).

Communicating with EWA involves the following steps:

- EWA will establish a TCP connection with your application (the server), EWA will attempt to connect to 127.0.0.1 and the port you provide to EWA
- Your server must send a “server accept” message to the client
- After receiving a HELO from the client, the server must reply with a code 250 message that includes the clients IP address, and gives a greeting message
- After receiving the MAIL FROM from the client, the server must reply with 250
- After receiving the RCPT TO from the client, the server must reply with 250
- After receiving the DATA command from the client, the server must reply with a 354 message if the name supplied can be used as a valid filename (this deviates from the SMTP protocol, see the protocol defined in the header file), or an error message if it cannot be used as a valid filename
- Any data received from the client should be stored in this file, data might come as one or more data streams, and ending with a CRLF CRLF . CRLF (a single dot on a separate line), when this “end of file” marker is received the file must be saved and closed by the server
- The client can then send either a QUIT command, or send a new file by sending a new DATA command
- When QUIT is received the server application should send a 221 message, and can then shut down the server application

The task should not be solved using Curl or other third-party libraries and should not be based on launching other applications in the operating system - only the use of Sockets as we have learned in lecture 10 on Network will give points on the task.

Required documentation:

You must take a screenshot of the program when it is running, the screenshot must be attached to the answer in an image file named task5_scrnshot.png (in the same folder as your makefile) AND must also be inserted in a suitable place in the text answer (PDF file). The screenshot must include both your running application, as well as the EWA tool after you have received a file (if you were able to do so). Any files received by the EWA tool must also be included in your exam answer, the files should have the filename as was sent as part of the DATA command and be stored in the same folder as your makefile (pg3401_exam25/task4_[candidate number]).

Task 6. Problem solving (20 %)

This task is intended as a more difficult task than the previous, and it will combine different techniques and functions with the need for some problem solving on your part. This task might use code that is part of the other tasks, so solving this task might be more difficult if you have not already completed the previous tasks.

For this task you must first do the “Required preparation” as described above. In the EWA make sure you have chosen 0 to create and initialize the folder structure, and have supplied your candidate number for this exam.

You will create an application that works as a network client, the EWA tool will bind to a port on the loopback adapter 127.0.0.1, and will listen on a random port, the port is displayed in the EWA tool when the port is ready to receive data.

Your application must accept the IP address and port number of the server as parameters from the terminal, for instance:

```
    ./task6 -server 127.0.0.1 -port 42420
```

When you connect to the EWA tool, first select option 6 in the main interface, and it will then send you a file, when creating the code for the application you first need to test the data you get and understand which protocol is used – neither this task nor the EWA prompt will tell you which protocol it is, you then need to create the code that uses this protocol correctly so the resulting code is able to receive the file sent.

The file received is in binary format, and the received file is encrypted with the TEA encryption method, where the encryption key is 128 bit but where each byte in the key is the same (so the key might be 0x11111111 11111111 11111111 11111111, 0x42424242 42424242 42424242 42424242, or similar. Your code needs to find the encryption key and then decrypt the file. The resulting file is in text format and consists only of 7-bit ASCII (and is readable English).

Finally, your code must save the file to disk, it should be saved in the same folder as your makefile.

Required documentation:

You must take a screenshot of the program when it is running, the screenshot must be attached to the answer in an image file named task6_scrnshot.png (in the same folder as your makefile) AND must also be inserted in a suitable place in the text answer (PDF file). Your application should print the encryption key found to the screen as part of your “debug” output, and this should be visible in said screenshot. The file received must also be included in the exam submission (in decrypted form).

+

End of task set