

The goal of this project is to create a C++ program that decodes a hidden message using synchronization and multithreading. You will also need to be comfortable with pointers.

These programs will decode a message from input given through STDIN and print the message to STDOUT. There are four test cases that need to pass in order for the project to be complete. Each test case file in the project zip folder has the STDIN input, compfile.txt, and what the correct output should be. Program will be run on Linux and should be one file only: main.cpp

In order to decode the message - you have to know what Fibonacci codes are:

This program will use what are called Fibonacci codes, which are simply codes that represent a number. For example the Fibonacci code for the number 6 is 10011. There is a Fibonacci code for every number. It is not important to understand the formula for calculating a Fibonacci code, but it is important to understand what a Fibonacci code represents. Every Fibonacci code is unique and must end with 11.

Here's a table of some Fibonacci codes for the numbers 1-10:

Symbol	Fibonacci representation	Fibonacci code word
1	$F(2)$	11
2	$F(3)$	011
3	$F(4)$	0011
4	$F(2) + F(4)$	1011
5	$F(5)$	00011
6	$F(2) + F(5)$	10011
7	$F(3) + F(5)$	01011
8	$F(6)$	000011
9	$F(2) + F(6)$	100011
10	$F(3) + F(6)$	010011

Included in the project zip is a file fibonacci.cpp

This file has a working program that will convert any integer to a fibonacci code.

In the program it is hardcoded to convert the number 7 to it's fibonacci code

You can use this code if you would like, but you don't have to.

SEE NEXT PAGE

Example Input to STDIN:

```
7
C 2
O 1
S 1
  1
3 2
6 1
0 1
compfile1.txt
```

- The data will always be inputted to STDIN in this format.
- The first number 7 represents the number of symbols/characters.
- The number next to each symbol represents the frequency of each character in the final decoded message
- The server program will assign a Fibonacci code to each symbol based on it's frequency and ASCII value. If two or more symbols have the same frequency, the symbol with a higher ASCII value has higher priority
- The final line of STDIN "compfile1.txt" is the "compressed" file name which is a file in the same directory as the client and server programs which contains a series of Fibonacci codes to be decoded for the final message to be outputted

Example compfile1.txt:

```
111011001111010110110110001110011
```

If you look more closely you can see the individual Fibonacci codes which correspond to the symbol given in STDIN

```
111011001111010110110110001110011
```

1 = 11 = C (ASCII value 67)
2 = 011 = 3 (ASCII value 53)
3 = 0011 = S (ASCII value 83)
4 = 1011 = O (ASCII value 79)
5 = 00011 = 6 (ASCII value 54)
6 = 10011 = 0 (ASCII value 48)
7 = 01011 = "SPACE" (ASCII value 32)

So after a priority is assigned to each symbol - The program can decode the final message:

```
C O S C      3 3 6 0
111011001111010110110110001110011
```

SEE NEXT PAGE

FINAL NOTES / SUMMARY:

After assigning a positive integer value (starting from 1) to the symbols in the sorted alphabet, your program must create a child thread per number of symbols in the alphabet. Each child thread will determine the Fibonacci code based on the received integer value from the main thread. After the child threads calculate the Fibonacci code, they will print the information about the symbol, its frequency, and the Fibonacci code, writing the Fibonacci code into a memory location available to the main thread. Finally, the main thread will use the codes generated by the child threads to decompress a file.

Each child thread will execute the following tasks:

- 1-Receive the integer value needed to calculate the Fibonacci code from the main thread.
- 2-Calculate the Fibonacci code.
- 3-Print the information about the symbol (symbol, frequency, Fibonacci code). You must use the output message provided in the example below.
- 4-Write the received information into a memory location accessible by the main thread.
- 5-Finish its execution.

Your program must print the information about the symbols based on their order in the input file. Therefore, you must use synchronization mechanisms to guarantee that child threads print the information about each symbol in the correct order.

After receiving the Fibonacci codes from the child threads, the main thread decompresses the contents of a file (sequence of bits represented as a string) and prints the decompressed message.

- You cannot use global variables.
- you should only place shared variables in critical sections, So try not to place print statements, etc. And technically they should only be in child threads.
- You must use POSIX threads. You cannot use anything other than the pthread library.
- You can only use named POSIX semaphores, pthreads mutex semaphores, or pthreads condition variables to achieve synchronization. Using pthread_join or sleep to synchronize your threads is not allowed.
- You cannot use different memory addresses to pass the information from the parent thread to the child threads.

about pthread_join:

it can't be used to synchronize your threads but that does not mean you can't use it all we need to call pthread_join as many times as we call pthread_create
don't call pthread_join in the same loop where we call pthread_create

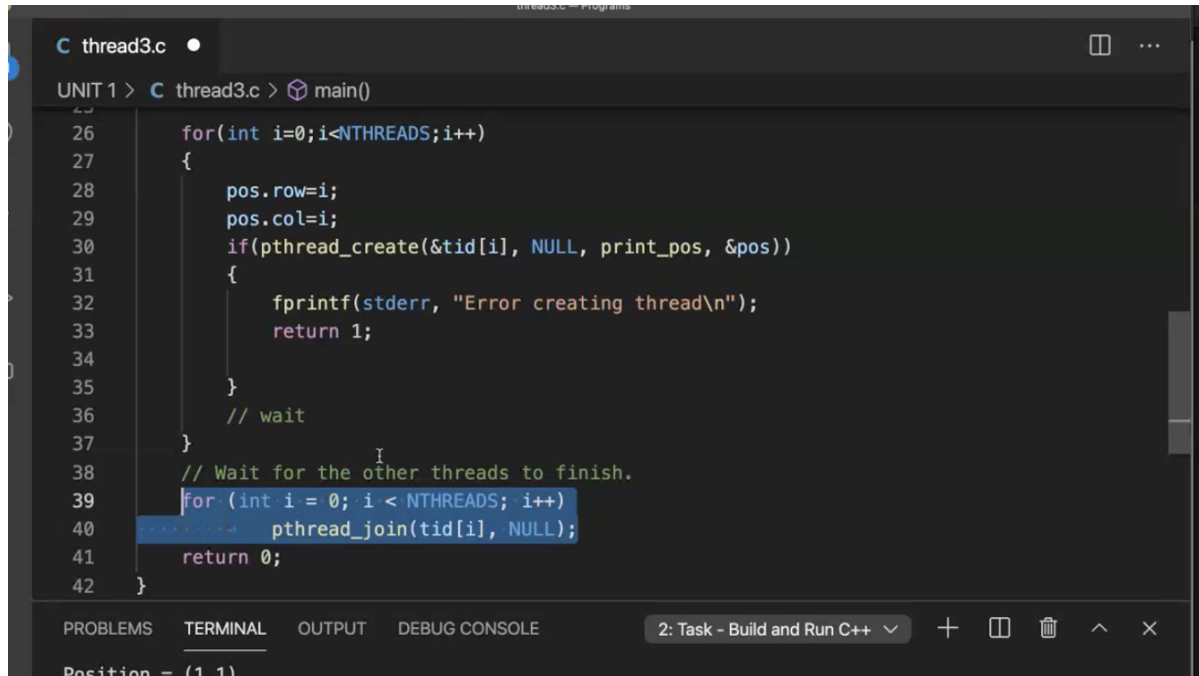
so you could have a for loop with pthread create and then another for loop with pthread join via

for example of how to use pthread_join - SEE NEXT PAGE:

EXAMPLE OF HOW TO USE pthread_join (GOOD WAY):

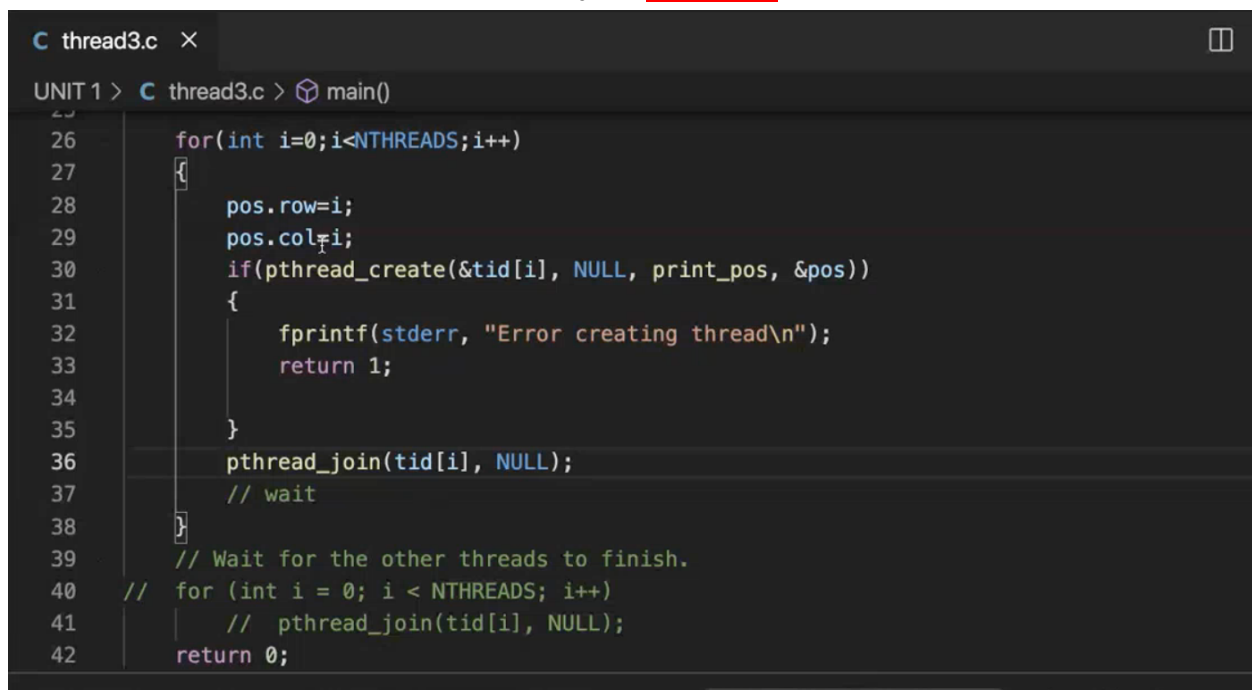
notice how pthread_join is in a separate loop

you should use pthread_join to guarantee that your parent thread will wait for all child threads before it ends. This is for proper multithreading.



```
UNIT 1 > C thread3.c > main()
26   for(int i=0;i<NTHREADS;i++)
27   {
28       pos.row=i;
29       pos.col=i;
30       if(pthread_create(&tid[i], NULL, print_pos, &pos))
31       {
32           fprintf(stderr, "Error creating thread\n");
33           return 1;
34       }
35       // wait
36   }
37   // Wait for the other threads to finish.
39   for (int i = 0; i < NTHREADS; i++)
40       pthread_join(tid[i], NULL);
41   return 0;
42 }
```

EXAMPLE OF HOW NOT TO USE pthread_join (BAD WAY - DO NOT DO THIS):



```
UNIT 1 > C thread3.c > main()
26   for(int i=0;i<NTHREADS;i++)
27   {
28       pos.row=i;
29       pos.col=i;
30       if(pthread_create(&tid[i], NULL, print_pos, &pos))
31       {
32           fprintf(stderr, "Error creating thread\n");
33           return 1;
34       }
35       pthread_join(tid[i], NULL);
36       // wait
37   }
38   // Wait for the other threads to finish.
40   // for (int i = 0; i < NTHREADS; i++)
41   //     pthread_join(tid[i], NULL);
42   return 0;
```