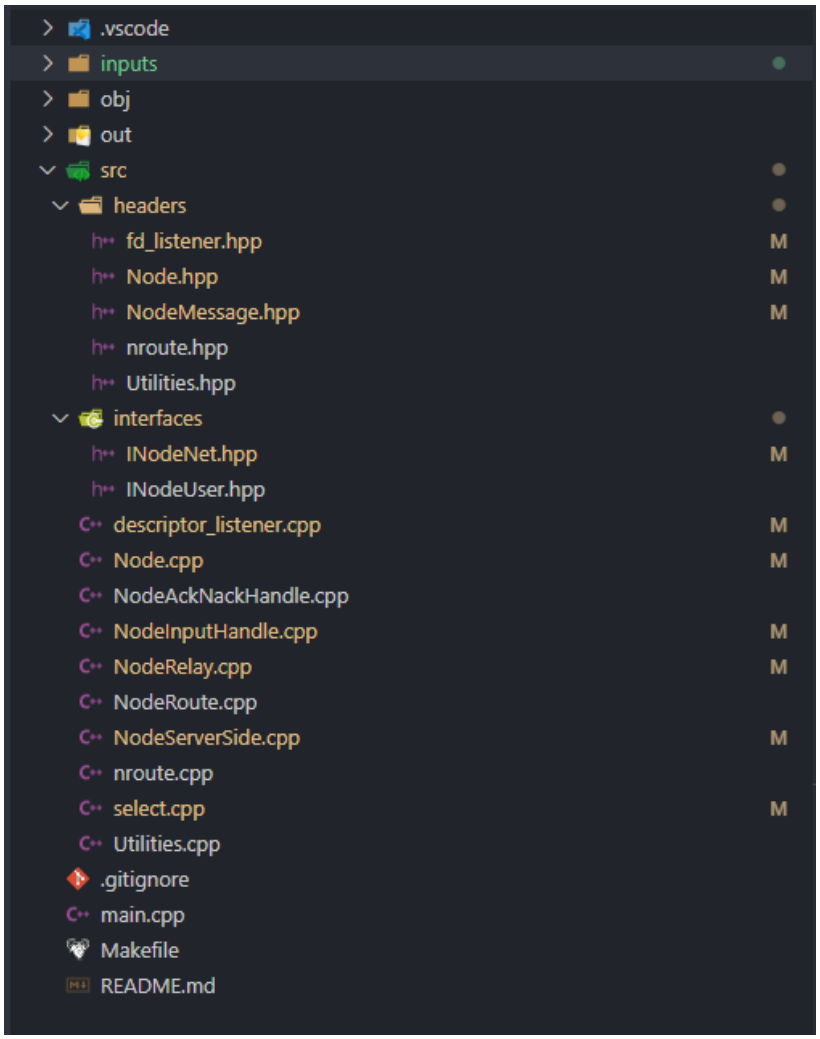
Communications project Part 3 - Readme

In this file we will do a brief overview on the project, and its implementation.
in general we had to build a "network" of nodes, that are following a protocol
from communication between nodes in the network.

we had to implement the protocol and make sure everything works.

First, note that:
in this implementation, the network can find nodes even if the network, have loops.
even tho we were asked to implement it on "three" graphs, only.

First how the project is structured:



this project is quite big so we have broke the project into lots of files,
as we should of course.

Node.hpp

the file Node.hpp, is basically the main class,
that has all the methods needed i.e call it a super class or whatever.

contains all the logic regarding connecting to nodes, creating server,
responding to messages by the protocol etc...

coresponding cpps:

every cpp containing the word Node, is implementing,
some part of Node class.

fd_listener.h

this class is a class that uses the select method, with gives us the ability to listen,
to different file descriptors at the same time (i.e listen to multiple sockets at once).

NodeMessage.h

the declarationg of NodeMessage stuct, with is infact the structure,
of all messages sent via the given protocol,
also have some other miner structs or "enums" with the corresponding values.

Utilities.h

contains some utility methods, to make life easier,
we also defined a Address, struct in here, with basically a convinient way, to save Addresses,
from string to string and port.

How to Use:

Surely with that project structure it might be "hard" to actually use the thing,
but worry not, as we have a **makefile** that do the job in a second.

basically our makefile is a modified version of my other makefile for Cpp projects,
that i have modified from stackoverflow, with let me compile everything in a given directory.
for this project i have modified the makefile to compile everything in the SRC folder.
and set the "header" and "interface" folder inside of it as "dependicies" i.e,
if something have changed inside of it we will compile everything.

So how to use the makefile:

```
run: $(EXE_PATH)$(PROG_NAME)
  ./$^ $(ip) $(port)

build: $(EXE_PATH)$(PROG_NAME)
  echo build completed
```

basically run:

```
make run
```

with would compile everything and run the exe file.
not this project works on linux only!

Example of running

to make life easier, the node will listen to user input,
and sockets inputs in two different threads.

why? because we want to use linux piping in order to substitude user input,
at least partially.

if you look closely we have the folder "inputs", this folder contains,
example of different inputs given to nodes.

first we have added "sof", "sleep", "eof"
to user input, "sof" simply means we are reading from file so repeat every line we read,
and print it to use log (so we can see what's happening),
the "eof" means the file has ended so now will start user input, and we dont want the node to repeat user input again,
the "sleep" simply tells the system to wait for a short duration.

for instance :

node10.txt

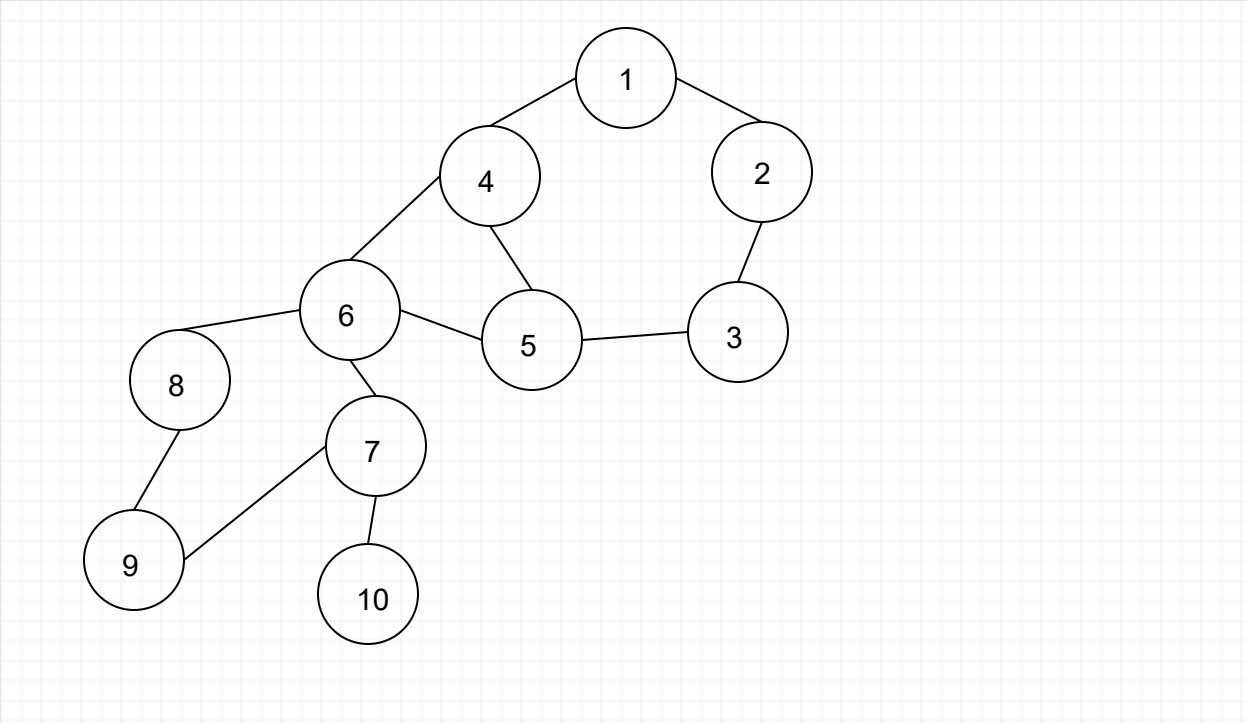
```
127.0.0.1
5010
sof
setid,10
sleep
connect,127.0.0.1:5007
sleep
route,2
route,3
route,333
sleep
send,2,40,tuturu
send,9,40,tutu
send,8,40,kuku
eof
```

so what's happening in this file,
basicaly we start a node, set the ip to "127.0.0.1" and the port to "5010"
then we set the node id to 10.
connect to node at port 5007 (with is node 7 for convience)
then we are trying to find the route to node 2, and node 3.
we then try to find route to 333 with doesnt exist.
then we send messages to 2,9,8.

how do we start this node?

```
cat inputs/node10.txt - | make run
```

the topology of the example from above:



clearly this topology contains cycles:
now lets look at the input and output of every node:

```
setid,1
Ack
New connection : 2
Message from : 2,
i am an automatic, test hehe...
Message from : 2,
automatic message number 2
Message from : 2,
test number 3
New connection : 4
Message from : 4,
tuturu
Message from : 4,
mayoshi is here
Message from : 9,
tuturu
```

```
setid,2
Ack
connect,127.0.0.1:5001
Connected to : 1
Ack
send,1,40,i am an automatic,
test hehe...
send,1,40,automatic message
number 2
send,1,40,test number 3
New connection : 3
peers
Ack
Peers :
1,3
Message from : 3,
test msg 333
Message from : 3,
test msg 2
Message from : 6,
tuturu
Message from : 10,
tuturu
```

```
setid,3
Ack
connect,127.0.0.1:5002
Connected to : 2
Ack
send,2,40,test msg 333
New connection : 5
send,2,40,test msg 2
peers
Ack
Peers :
2,5
Message from : 5,
tuturu
Message from : 6,
tuturu
```

```
setid,4
Ack
connect,127.0.0.1:5001
Connected to : 1
Ack
send,1,40,tuturu
send,1,40,mayoshi is here
peers
Ack
Peers :
1
New connection : 5
New connection : 6
Message from : 5,
tuturu
Message from : 6,
tuturu
```

```
setid,5
Ack
connect,127.0.0.1:5003
Connected to : 3
Ack
sleep
New connection : 6
connect,127.0.0.1:5004
Connected to : 4
Ack
sleep
send,3,40,tuturu
Ack
send,4,40,tuturu
peers
Ack
Peers :
3,6,4
```

```
setid,6
Ack
connect,127.0.0.1:5005
Connected to : 5
Ack
sleep
connect,127.0.0.1:5004
Connected to : 4
Ack
sleep
send,3,40,tuturu
Ack
send,4,40,tuturu
Ack
New connection : 7
send,2,40,tuturu
Ack
New connection : 8
peers
Ack
Peers :
5,4,7,8
```

```
setid,7
Ack
connect,127.0.0.1:5006
Connected to : 6
Ack
peers
Ack
Peers :
6
New connection : 9
New connection : 10
```

```
setid,8
Ack
connect,127.0.0.1:5006
Connected to : 6
Ack
peers
Ack
Peers :
6
New connection : 9
Message from : 10,
kuku
```

```
setid,9
Ack
connect,127.0.0.1:5008
Connected to : 8
Ack
connect,127.0.0.1:5007
Connected to : 7
Ack
sleep
route,1
Ack
7->6->4->1
sleep
send,1,40,tuturu
Ack
Message from : 10,
tutu
```

```
setid,10
Ack
sleep
connect,127.0.0.1:5007
Connected to : 7
Ack
sleep
route,2
Ack
7->6->5->3->2
route,3
Ack
7->6->5->3
route,333
Nack
sleep
send,2,40,tuturu
Ack
send,9,40,tutu
Ack
send,8,40,kuku
Ack
```

Clearly this test shows that's everything works as intended.

Another test for removing nodes, and trying to send message again.
some test on what will happen on deleting nodes:
in this example we see the output/input of node 6.

we try to send data to node 1 and 2, while in the middle we deleted node 4.
notice: (time s) means how much time the route is alive. (after 60 sec we delete the route).

```
time 3
Ack
4->1->2
route,2
time 5
Ack
4->1->2
route,2
time 7
Ack
4->1->2
send,2,5,kek2
time 14
Ack
send,1,10,kekk
time 0
Ack
send,2,10,kekke
time 40
Ack
send,1,5,test
time 27
Ack
route,1
time 6
Ack
5->3->2->1
```

clearly the nodes can bypass the hassle of removing nodes,
there might be some minor bugs but generally after 60 seconds any bug in the system should resolve, as all routes would update at least once.

Thanks for reading