Artificial Intelligence (CS 701)

Lab Report Submitted to

Indian Institute of Information Technology Surat

for

Bachelor of Technology

In

Electronics and Communications Engineering Department

Submitted by

Ashish Lathkar (UI19EC07)

Nishant Singh (UI19EC28)

Course Faculty

Dr. Pradeep Kumar Roy

Ms. Shraddha PateL

Department of Computer Science and Engineering

Indian Institute of Information Technology Surat

Gujarat-395007,India

Experiment – 1

Program Name: Weather Chatbot (Crazy)

Description:

This version of an Artificial Intelligence Chatbot directly interacts with users through the terminal, giving weather-related information for all cities in India. However, the user has to be specific in their questions as the AI bot is still very "young", as it calls itself and is still learning to comprehend and provide correct answers to questions.

With the greater availability of AI libraries, particularly with the given dataset's support, Chatbots can now automatically take actions pursuant to user inputs. The result is a whole new way for users to more conveniently communicate their intents and for systems to automatically recognize and act on these intents.

In finance, E-commerce and other industries, chatbots have become increasingly valuable for helping consumers figure out what they need and get it promptly. Chatbots help to simulate the scenario of talking with a human being while a machine is getting most of the work done. Big companies around the globe, like Google and Facebook, have already begun experimenting with chatbots, and this could have an effect in transforming the way that the Internet is seen and used, especially for disaster response and preparedness in terms of providing services such as weather information that a machine could more effectively handle. A challenge that needs to be overcome is to make the interaction between the human and the device (Chatbot) easy for the human to grasp for it to be valid.

Data Collection:

For now, we trained our chatbot using a JSON format dataset. But we are trying to make it more dynamic using OpenWeatherMap API.

Steps Involved in Building ChatBot:

1. Setting up our project (installing required modules)
2. Creating intents in JSON format
3. Training and saving ChatBot
4. User interaction

Defining the Intentions of a ChatBot:

We need to define a few simple intents and a group of messages that match those intents and also map some responses based on each intent category. First, I'll create a JSON file named "intents.json", including this data as follows:

```
{"intents": [
   {"tag": "greeting",
    "patterns": ["Hi", "Hey", "Is anyone there?", "Hello", "Hay"],
    "responses": ["Hello", "Hi", "Hi there"]
   },
   {"tag": "goodbye",
    "patterns": ["Bye", "See you later", "Goodbye"],
    "responses": ["See you later", "Have a nice day", "Bye! Come back again"]
   },
   {"tag": "about",
    "patterns": ["Who are you?", "What are you?", "Who you are?" ],
    "responses": ["I'm Crazy, your bot assistant", "I'm Crazy, an Artificial Intelligent bot"]
   },
   {"tag": "name",
   "patterns": ["what is your name", "what should I call you", "whats your name?"],
   "responses": ["You can call me Crazy.", "I'm Crazy!", "Just call me as Crazy"]
   },
   {"tag": "help",
   "patterns": ["Could you help me?", "give me a hand please", "Can you help?", "What can you do for
me?", "I need a support", "I need a help", "support me please"],
   "responses": ["Tell me how can assist you", "Tell me your problem to assist you", "Yes Sure, How can I
support you"]
   },
   {"tag": "weather",
   "patterns": ["tell me about weather", "weather report", "what about weather"],
   "responses": ["Yes, tell me for which city you like to check weather?", "name the city"]
   },
   {"tag": "city1",
   "patterns": ["surat", "what about surat", "surat climate"],
   "responses": ["\nFor Surat,\nTemperature: 31C\nPrecipitation: 0 percent\nHumidity: 27 percent\nWind:
11 km/h"]
   },
   {"tag": "city2",
   "patterns": ["mumbai", "what about mumbai", "mumbai climate"],
   "responses": ["\nFor Mumbai,\nTemperature: 31C\nPrecipitation: 0 percent\nHumidity: 33
percent\nWind: 16 km/h"]
   },
```

Flow:

1. Installing Required Libraries

```
import json
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, GlobalAveragePooling1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
```

- **JSON –** able to import and parse JSON data into a Google sheets table.
- **Numpy –** to perform a wide variety of mathematical operations
- **Tensorflow –** to create deep learning models directly or fast numerical computing
- **Keras –** an open-source neural network library that runs on top of tensorflow
- **Why sequential –** model has multiple inputs or multiple outputs + need to do layer sharing + want non-linear topology
- **Tokenizer –** breaking the raw text into small chunks and into words, sentences called tokens. Help in understanding the context and developing the model for the NLP. (Help in interpreting the meaning of the text by analyzing the sequence of the words)
- **Pad_Sequences –** to ensure that all sequences in a list have the same length. By default, this is done by padding 0 in the beginning of each sequence until each sequence has the same length as the longest sequence.
- **LabelEncoder –** can be used to normalize labels. Transform non-numerical labels to numerical labels and return encoded labels.

2. Loading the Dataset and Process the Required Files

```
with open('intents.json') as file:
    data = json.load(file)

training_sentences = []
training_labels = []
labels = []
responses = []


for intent in data['intents']:
    for pattern in intent['patterns']:
        training_sentences.append(pattern)
        training_labels.append(intent['tag'])
    responses.append(intent['responses'])

    if intent['tag'] not in labels:
        labels.append(intent['tag'])

num_classes = len(labels)
```

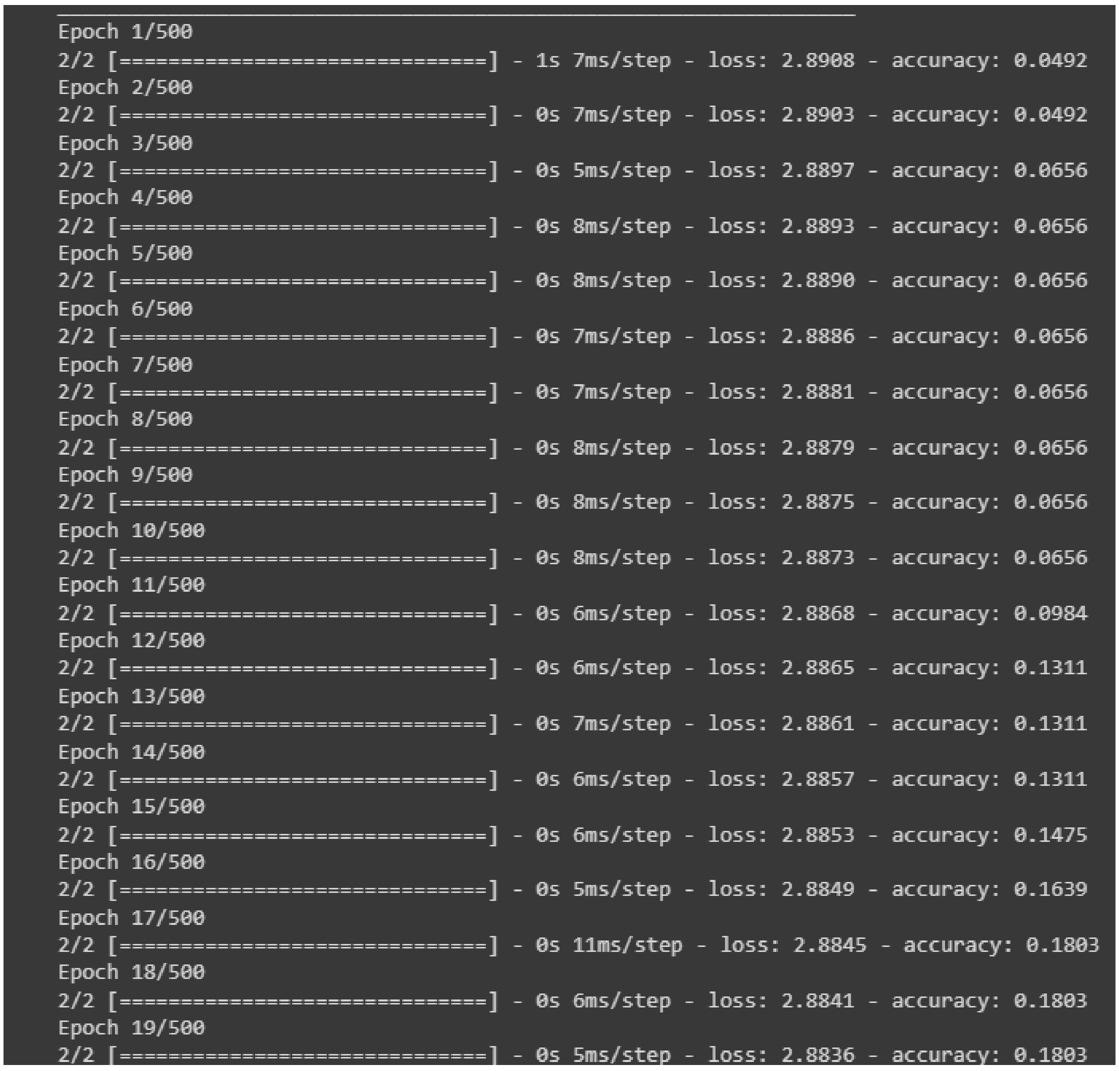
```
lbl_encoder = LabelEncoder()
lbl_encoder.fit(training_labels)
training_labels = lbl_encoder.transform(training_labels)
```

3. Working of Tokenizer and Pad_Sequences (vectorize the data)

```
vocab_size = 1000
embedding_dim = 16
max_len = 20
oov_token = "<OOV>"

tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_token)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded_sequences = pad_sequences(sequences, truncating='post', maxlen=max_le
n)
```

4. Modelling and Training a Neural Network

```
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
model.add(GlobalAveragePooling1D())
model.add(Dense(16, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])

model.summary()
epochs = 500
history = model.fit(padded_sequences, np.array(training_labels), epochs=epoc
hs)
```



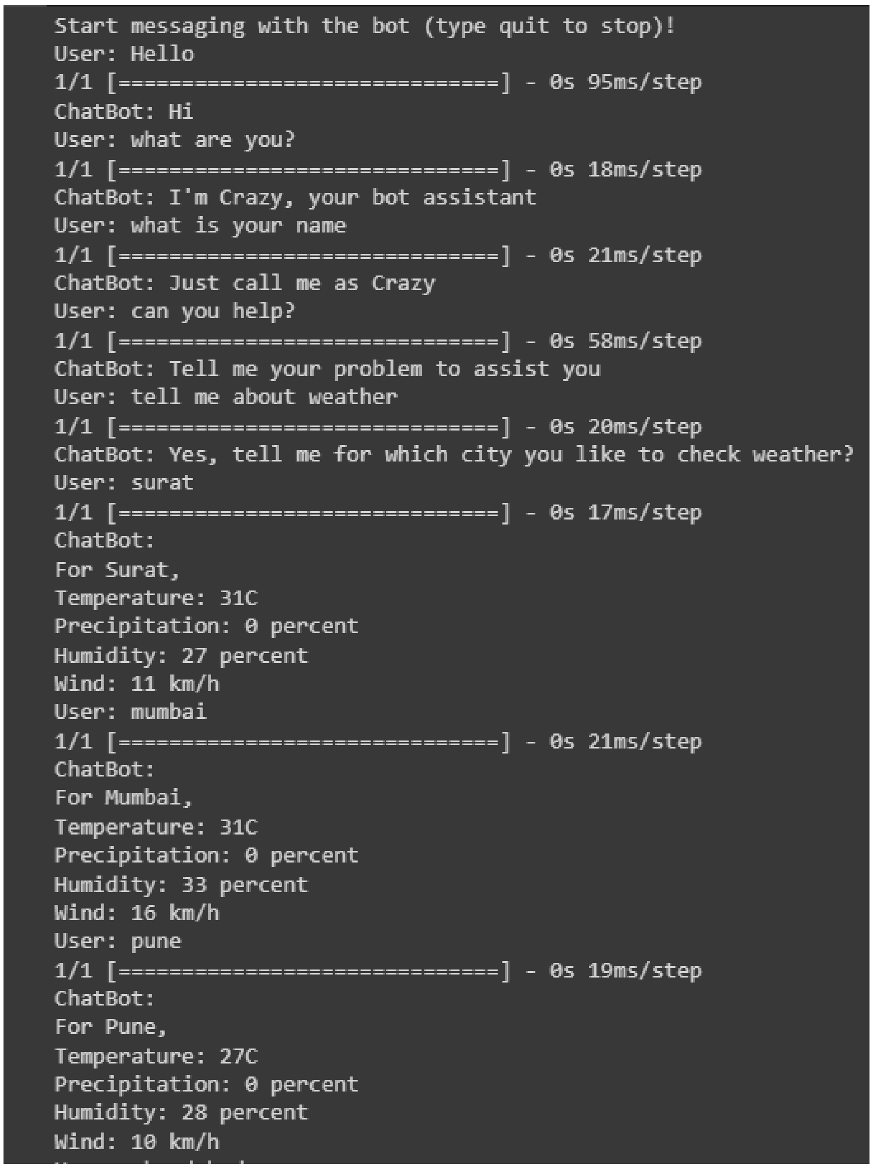
```
Epoch 1/500
2/2 [==============================] - 1s 7ms/step - loss: 2.8908 - accuracy: 0.0492
Epoch 2/500
2/2 [==============================] - 0s 7ms/step - loss: 2.8903 - accuracy: 0.0492
Epoch 3/500
2/2 [==============================] - 0s 5ms/step - loss: 2.8897 - accuracy: 0.0656
Epoch 4/500
2/2 [==============================] - 0s 8ms/step - loss: 2.8893 - accuracy: 0.0656
Epoch 5/500
2/2 [==============================] - 0s 8ms/step - loss: 2.8890 - accuracy: 0.0656
Epoch 6/500
2/2 [==============================] - 0s 7ms/step - loss: 2.8886 - accuracy: 0.0656
Epoch 7/500
2/2 [==============================] - 0s 7ms/step - loss: 2.8881 - accuracy: 0.0656
Epoch 8/500
2/2 [==============================] - 0s 8ms/step - loss: 2.8879 - accuracy: 0.0656
Epoch 9/500
2/2 [==============================] - 0s 8ms/step - loss: 2.8875 - accuracy: 0.0656
Epoch 10/500
2/2 [==============================] - 0s 8ms/step - loss: 2.8873 - accuracy: 0.0656
Epoch 11/500
2/2 [==============================] - 0s 6ms/step - loss: 2.8868 - accuracy: 0.0984
Epoch 12/500
2/2 [==============================] - 0s 6ms/step - loss: 2.8865 - accuracy: 0.1311
Epoch 13/500
2/2 [==============================] - 0s 7ms/step - loss: 2.8861 - accuracy: 0.1311
Epoch 14/500
2/2 [==============================] - 0s 6ms/step - loss: 2.8857 - accuracy: 0.1311
Epoch 15/500
2/2 [==============================] - 0s 6ms/step - loss: 2.8853 - accuracy: 0.1475
Epoch 16/500
2/2 [==============================] - 0s 5ms/step - loss: 2.8849 - accuracy: 0.1639
Epoch 17/500
2/2 [==============================] - 0s 11ms/step - loss: 2.8845 - accuracy: 0.1803
Epoch 18/500
2/2 [==============================] - 0s 6ms/step - loss: 2.8841 - accuracy: 0.1803
Epoch 19/500
2/2 [==============================] - 0s 5ms/step - loss: 2.8836 - accuracy: 0.1803
```

5. Creating Functions and Saving Model

```
# to save the trained model
model.save("chat_model")

import pickle

# to save the fitted tokenizer
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

# to save the fitted label encoder
with open('label_encoder.pickle', 'wb') as ecn_file:
    pickle.dump(lbl_encoder, ecn_file, protocol=pickle.HIGHEST_PROTOCOL)
```

```
!pip install colorama
import colorama
colorama.init()
from colorama import Fore, Style, Back

import random
import pickle

with open("intents.json") as file:
    data = json.load(file)


def chat():

    # load trained model
    model = keras.models.load_model('chat_model')

    # load tokenizer object
    with open('tokenizer.pickle', 'rb') as handle:
        tokenizer = pickle.load(handle)

    # load label encoder object
    with open('label_encoder.pickle', 'rb') as enc:
        lbl_encoder = pickle.load(enc)

    # parameters
    max_len = 20

    while True:
        print(Fore.LIGHTBLUE_EX + "User: " + Style.RESET_ALL, end="")
        inp = input()
        if inp.lower() == "quit":
            break
        result = model.predict(keras.preprocessing.sequence.pad_sequences(to
kenizer.texts_to_sequences([inp]),
                                             truncating='post', maxlen=max_l
en))
        tag = lbl_encoder.inverse_transform([np.argmax(result)])

        for i in data['intents']:
            if i['tag'] == tag:
                print(Fore.GREEN + "ChatBot:" + Style.RESET_ALL , np.random.
choice(i['responses']))

print(Fore.YELLOW + "Start messaging with the bot (type quit to stop)!" + St
yle.RESET_ALL)
chat()
```

It’s Time to See Output:



```
Start messaging with the bot (type quit to stop)!
User: Hello
1/1 [==============================] - 0s 95ms/step
ChatBot: Hi
User: what are you?
1/1 [==============================] - 0s 18ms/step
ChatBot: I'm Crazy, your bot assistant
User: what is your name
1/1 [==============================] - 0s 21ms/step
ChatBot: Just call me as Crazy
User: can you help?
1/1 [==============================] - 0s 58ms/step
ChatBot: Tell me your problem to assist you
User: tell me about weather
1/1 [==============================] - 0s 20ms/step
ChatBot: Yes, tell me for which city you like to check weather?
User: surat
1/1 [==============================] - 0s 17ms/step
ChatBot:
For Surat,
Temperature: 31C
Precipitation: 0 percent
Humidity: 27 percent
Wind: 11 km/h
User: mumbai
1/1 [==============================] - 0s 21ms/step
ChatBot:
For Mumbai,
Temperature: 31C
Precipitation: 0 percent
Humidity: 33 percent
Wind: 16 km/h
User: pune
1/1 [==============================] - 0s 19ms/step
ChatBot:
For Pune,
Temperature: 27C
Precipitation: 0 percent
Humidity: 28 percent
Wind: 10 km/h
```

Future Work:

In the future, we plan to improve the model using a dynamics weather application. We can integrate several APIs available under OpenWeatherMap in the model. It will be able to fetch weather reports for any place on the global level. We would like to include more languages for high reach. And also, we are in search of more authentic datasets regarding the weather in India. We are also looking forward to developing an android application.

Conclusion:

Weather chatbots have lots of importance regarding educational or agricultural purposes. This will help to find the queries for people to judge the weather and plan actions accordingly. It has several impactful benefits, such as cost savings, offering website visitors context, better analysis of customer data, and enhanced customer engagement and sales. In addition, bots save a great deal of time. We will make it into an app so everyone can use it easily.

Experiment – 2

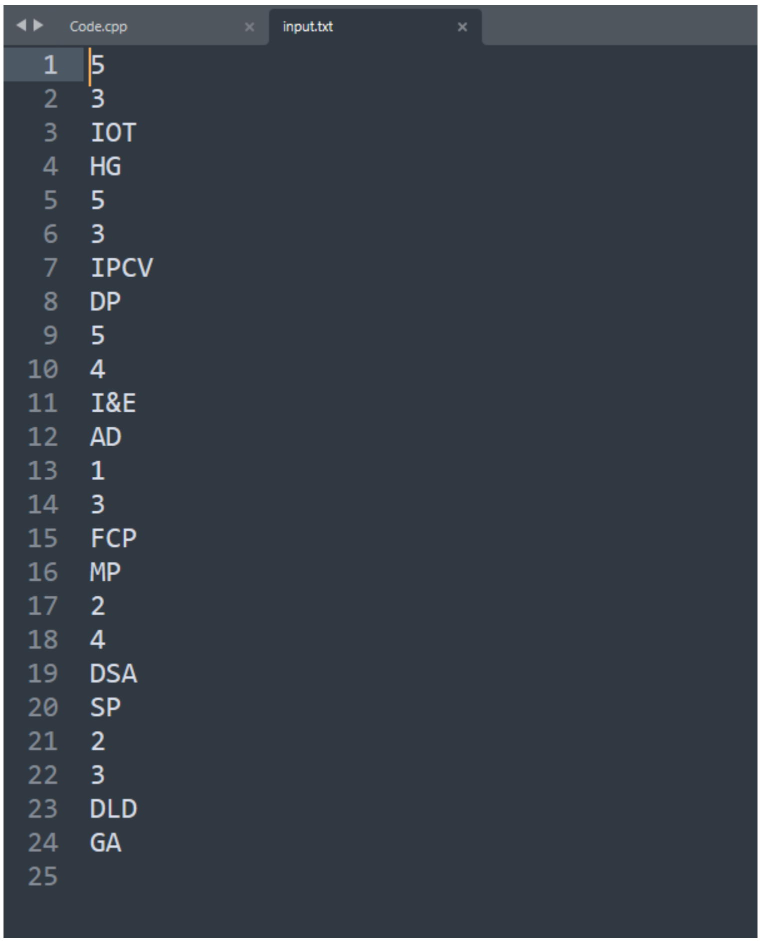
Aim: To create algorithm which can generate timetable for a college provided no. of subject, semester, classes, faculty etc.

Code:

```
#include<bits/stdc++.h>
#include <iostream>
#include <fstream>
using namespace std;

class Node
{
public:
  string semester = "1", numberOfLectures = "1";
  string name = "", intial = "", subject = "";

  Node() {}
};

class GenerateTimeTable
{
private:
  int classrooms, workingDays, totalSlots;
  vector<vector<vector<Node>>> tt, ans;

  vector<string> weekdays = {"Monday", "Tuesday", "Wednesday", "Thrusday", "Friday"};
  vector<string> slot = {"09:30 - 10:30", "10:30 - 11:30", "11:30 - 12:30", "14:30 - 15:30", "15:30 -
16:30", "16:30 - 17:30"};

  int stringTointeger(string str)
  {
    int temp = 0;
    for (int i = 0; i < str.length(); i++) {
      temp = temp * 10 + (str[i] - '0');
    }
    return temp;
  }
public:
  GenerateTimeTable(int classrooms=6, int workingDays=5, int totalSlots=6)
  {
    this->classrooms = classrooms;
    this->workingDays = workingDays;
    this->totalSlots = totalSlots;
    tt.resize(classrooms, vector<vector<Node>>(workingDays, vector<Node>(totalSlots)));
  }
  bool isValid(int x, int y, int z, Node e)
  {
    // check that on the same working day that professor/teacher must not be teaching on that slot in
any other classroom
    for (int i = 0; i < x; i++) {
      if(i==x)
        continue;
      if((e.name == tt[i][y][z].name))
        return false;

    }

    return true;
  }

  void fillTimeTable(int e, vector<Node> &entity)
  {
    if(e==entity.size()) {
      ans = tt;
      return ;
    }
    int lectures = stringTointeger(entity[e].numberOfLectures);

    for (int i = 0; i < classrooms; i++) {
      for (int j = 0; j < workingDays; j++) {
        for (int k = 0; k < totalSlots; k++) {
          if(tt[i][j][k].name== "" and isValid(i, j, k, entity[e])) {
            lectures -= 1;

            tt[i][j][k] = entity[e];
            if(lectures==0)
              fillTimeTable(e + 1, entity);
            else
              break;
          }
        }
      }
    }
  }

  vector<vector<vector<Node>>> getTimeTable()
  {
    return tt;
  }

  void printTimeTable()
  {
    for (int i = 0; i < classrooms; i++) {

      cout << "classroom : " << i + 1 << "\n";

      for (int j = 0; j < workingDays; j++) {
        cout << weekdays[j];
        cout << "\n";

        for (int k = 0; k < totalSlots; k++)
```

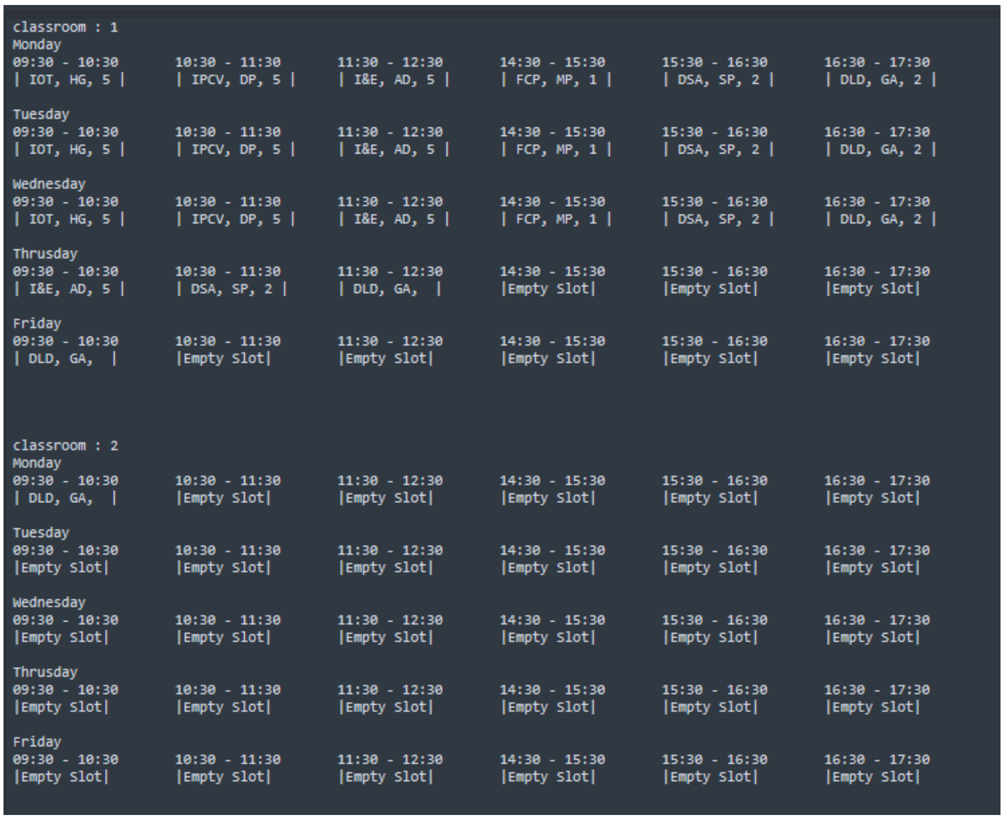
```
            cout << slot[k] << "\t\t";
            cout << "\n";

            for (int k = 0; k < totalSlots; k++) {
              if(ans[i][j][k].subject=="") {
                cout << "|Empty Slot|\t\t";
              }
              else
              cout << "| " << ans[i][j][k].subject << ", " << ans[i][j][k].name << ", " << ans[i][j][k].semester
<< " |\t\t";
            }
            cout << "\n\n";
          }

          cout << "\n\n\n";
        }
      }

};


int main() {
  vector<Node> entity;

  ifstream myfile;
  myfile.open("input.txt");

  if(myfile.is_open()) {
    string sem = "", nol = "", name = "", intial = "", subject = "";
    while (myfile)
    {
      getline(myfile, sem);
      getline(myfile, nol);
      getline(myfile, subject);
      getline(myfile, name);

      Node e = Node();
      e.name = name; e.subject = subject; e.intial = intial; e.semester = sem; e.numberOfLectures = nol;

      // cout << e.semester << endl;
      // cout << e.numberOfLectures << endl;
      // cout << e.name << endl;
      // cout << e.subject << endl;
      // cout << e.intial << endl;

      entity.push_back(e);
    }
  }

  GenerateTimeTable tt = GenerateTimeTable();
  tt.fillTimeTable(0, entity);
  tt.printTimeTable();

  return 0;
}
```

Input:



```
5
3
IOT
HG
5
3
IPCV
DP
5
4
I&E
AD
1
3
FCP
MP
2
4
DSA
SP
2
3
DLD
GA

```

Output:



```
classroom : 1
Monday
09:30 - 10:30       10:30 - 11:30       11:30 - 12:30       14:30 - 15:30       15:30 - 16:30       16:30 - 17:30
| IOT, HG, 5 |      | IPCV, DP, 5 |     | I&E, AD, 5 |      | FCP, MP, 1 |      | DSA, SP, 2 |      | DLD, GA, 2 |

Tuesday
09:30 - 10:30       10:30 - 11:30       11:30 - 12:30       14:30 - 15:30       15:30 - 16:30       16:30 - 17:30
| IOT, HG, 5 |      | IPCV, DP, 5 |     | I&E, AD, 5 |      | FCP, MP, 1 |      | DSA, SP, 2 |      | DLD, GA, 2 |

Wednesday
09:30 - 10:30       10:30 - 11:30       11:30 - 12:30       14:30 - 15:30       15:30 - 16:30       16:30 - 17:30
| IOT, HG, 5 |      | IPCV, DP, 5 |     | I&E, AD, 5 |      | FCP, MP, 1 |      | DSA, SP, 2 |      | DLD, GA, 2 |

Thrusday
09:30 - 10:30       10:30 - 11:30       11:30 - 12:30       14:30 - 15:30       15:30 - 16:30       16:30 - 17:30
| I&E, AD, 5 |      | DSA, SP, 2 |      | DLD, GA,  |       |Empty Slot|        |Empty Slot|        |Empty Slot|

Friday
09:30 - 10:30       10:30 - 11:30       11:30 - 12:30       14:30 - 15:30       15:30 - 16:30       16:30 - 17:30
| DLD, GA,  |       |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|



classroom : 2
Monday
09:30 - 10:30       10:30 - 11:30       11:30 - 12:30       14:30 - 15:30       15:30 - 16:30       16:30 - 17:30
| DLD, GA,  |       |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|

Tuesday
09:30 - 10:30       10:30 - 11:30       11:30 - 12:30       14:30 - 15:30       15:30 - 16:30       16:30 - 17:30
|Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|

Wednesday
09:30 - 10:30       10:30 - 11:30       11:30 - 12:30       14:30 - 15:30       15:30 - 16:30       16:30 - 17:30
|Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|

Thrusday
09:30 - 10:30       10:30 - 11:30       11:30 - 12:30       14:30 - 15:30       15:30 - 16:30       16:30 - 17:30
|Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|

Friday
09:30 - 10:30       10:30 - 11:30       11:30 - 12:30       14:30 - 15:30       15:30 - 16:30       16:30 - 17:30
|Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|        |Empty Slot|
```