Axel AOUIZERATE

Bamlak GURARA

Sami BOUALAMI

PROJECT C

Goals of the project :

- Store the words of the dictionary in a data structure
- Realize an algorithm to generate a random sentence
- Make a code that is efficient in memory and time

Language : C

Data structure used : ternary trees and head tail lists

Outline :

I. Introduction

The goal of this project is to create a program that automatically generates sentences that are grammatically and (as far as possible) orthographically correct, but don't necessarily make sense. This project uses a dictionary file of approximately 300,000 words from the French language. We have done this project in groups of 3. We used this advantage to share our different ideas, and help each other to make the functions as good as we can.

The instructions of the project were the following :

Firstly, we had to store all the words in a tree structure, and organize them, given their type (noun, adjective, verb or adverb), genre, plurality, and conjugation (for verbs). Also, the words were divided into two categories : base forms and inflicted forms. Each inflicted form is associated with one base form, which is in the same word family. For this matter, we stored the inflicted form in a structure contained in the corresponding base form.

Then, we had to do multiple functions with the dictionary : research words, return the genre and plurality of a given word, generate random sentences, perform automatic completion, etc. This report provides a brief overview of what we have accomplished since the beginning of the project. Furthermore, the difficulties we faced and the solutions that we found will be discussed. Finally, we will expose the main learning outcomes and our satisfactions concerning our work.

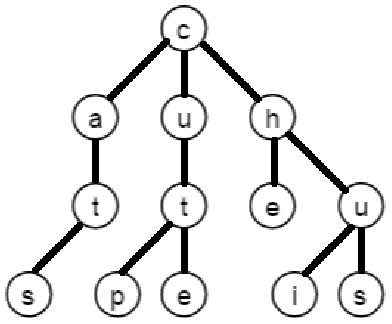
Here is a table showing which functions we have done :

Store the base forms in a tree (nouns, adjectives, adverbs, verbs)	Done
Store the inflicted forms in a structure	Done
Extract a random base form	Done
Search for a base form	Done
Search for an inflicted form, and return its genre, etc.	Not done
Generate random sentences :	Done
Model 1 for base forms	Done
Model 2 for base forms	Done
Model 1 for inflicted forms	Done
Model 2 for inflicted forms	Done
Option : Generate sentences with personal pronouns	Not done
Option : Automatic completion	Not done
Option : Add a missing inflicted form	Not done

Part 1 : data structure

1 - Define an adapted data structure :

We have chosen to store the **inflicted forms** in a **ternary tree**. This structure is pointed by the node which contains the last letter of the base form. This node also contains the integer counting the number of inflicted forms.



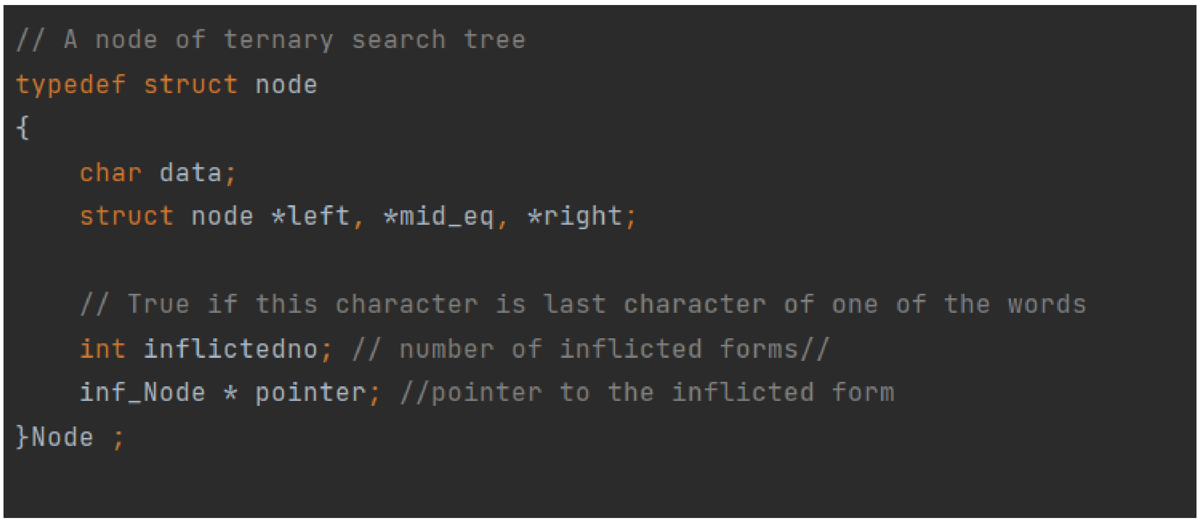
Ternary tree scheme

At first, we wanted to do an 26-node tries n-ary tree, so as to have a tree that is perfectly ordered with the letters closer to A on the left, and the letters close to Z on the right. We knew we couldn't do a 26 power 26 nodes tree because it would leave so many unused nodes, which is memory inefficient. So then, we tried to do another n-ary tree, with each node that would have a different number of sons depending on the words we want to implement after the node. In this way, we would have had to modify the node structure in the middle of the code, which we don't know how to do. Finally, the best alternative that we found, for both memory and simplicity of usage is the **ternary search tree**.

2 - Why did we chose the ternary tree ?

This tree has many advantages :

Firstly, it facilitates words research, thanks to the node structure below :



```
// A node of ternary search tree
typedef struct node
{
    char data;
    struct node *left, *mid_eq, *right;

    // True if this character is last character of one of the words
    int inflictedno; // number of inflicted forms//
    inf_Node * pointer; //pointer to the inflicted form
}Node ;
```

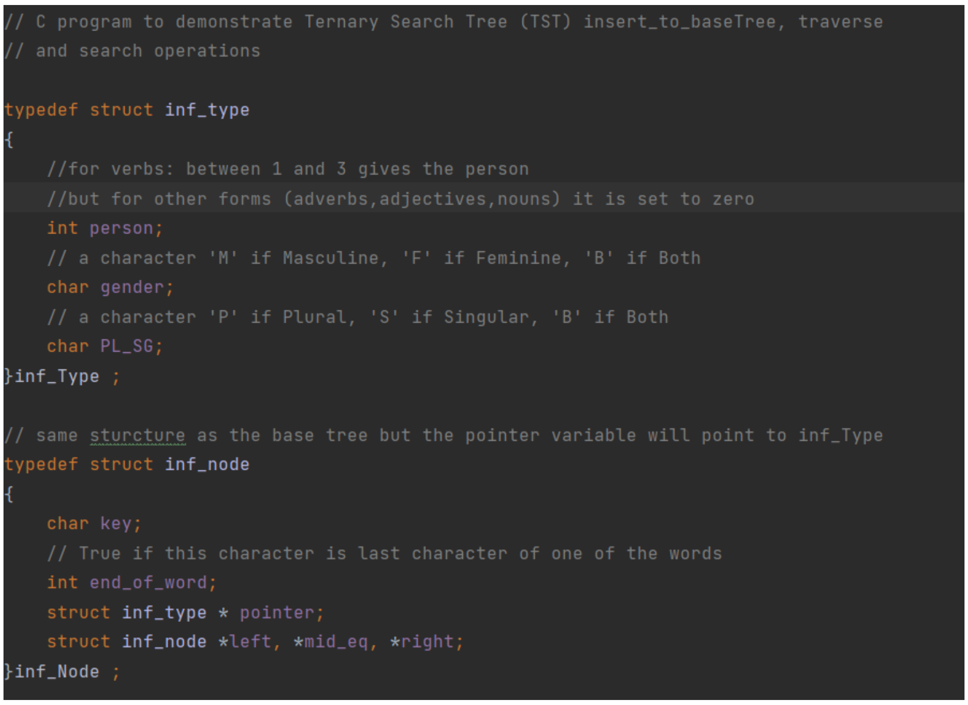
Each node points towards 3 nodes. 'left' node, 'right' node, and a node below, called 'eq'. When going through the tree, going to any of the 3 directions will have a different function. Going below will retain the letter contained in the previous node whereas going right or left doesn't retain the letter, as it is made to search for a given letter in the tree.

The Left pointer is a pointer to the node whose character is less than the value in the current node while the Right pointer is pointer to the node whose character is greater than the value in the current node. The equal/mid pointer points to the node whose value is equal to the current node value.

Also, the ternary tree it is very **efficient in memory**. Indeed, it doesn't waste memory because there is no unused node in the tree.

3 – Why should we stock a pointer towards a structure in a leaf node ?

Since only the leaf nodes contain the inflicted forms, it is better only to stock a pointer to the structure storing inflicted form. Indeed, they use only 4 byts of memory. That's a lot less that a structure like the ternary tree. Then, the nodes that do not contain the inflicted forms will contain pointers rather than heavier data structures. **Therefore, the use of pointers saves memory.** As we can see in the structure above, the inflicted forms tree is pointed by the leaf node.



```c
// C program to demonstrate Ternary Search Tree (TST) insert_to_baseTree, traverse
// and search operations

typedef struct inf_type
{
    //for verbs: between 1 and 3 gives the person
    //but for other forms (adverbs,adjectives,nouns) it is set to zero
    int person;
    // a character 'M' if Masculine, 'F' if Feminine, 'B' if Both
    char gender;
    // a character 'P' if Plural, 'S' if Singular, 'B' if Both
    char PL_SG;
}inf_Type ;

// same sturcture as the base tree but the pointer variable will point to inf_Type
typedef struct inf_node
{
    char key;
    // True if this character is last character of one of the words
    int end_of_word;
    struct inf_type * pointer;
    struct inf_node *left, *mid_eq, *right;
}inf_Node ;
```

This is the definition of the **inf_node** contained in the inflicted forms tree. As we can see, the leaf nodes of the inflicted tree contain a structure called **inf_type** indicating the genre, and the plurality of the word.

II – Explanation of the functions

We are going to explain our process of thinking for implementing all the functions below. Also, we will show the **screenshots** indicating that our functions are working :

1 - How to extract data from a .txt file

2 - How to construct the tree containing the nouns, adjectives, etc.

3 - Menu description

4 - Base form research

5 - Extract a random base form

6 - Generate a random sentence

  A - for base forms

  B - for inflicted form

  C – with personal pronouns (optional)

7 – different options

  A – automatic completion

  B – Implement a missing inflicted form

1 - How to extract data from a .txt file ?

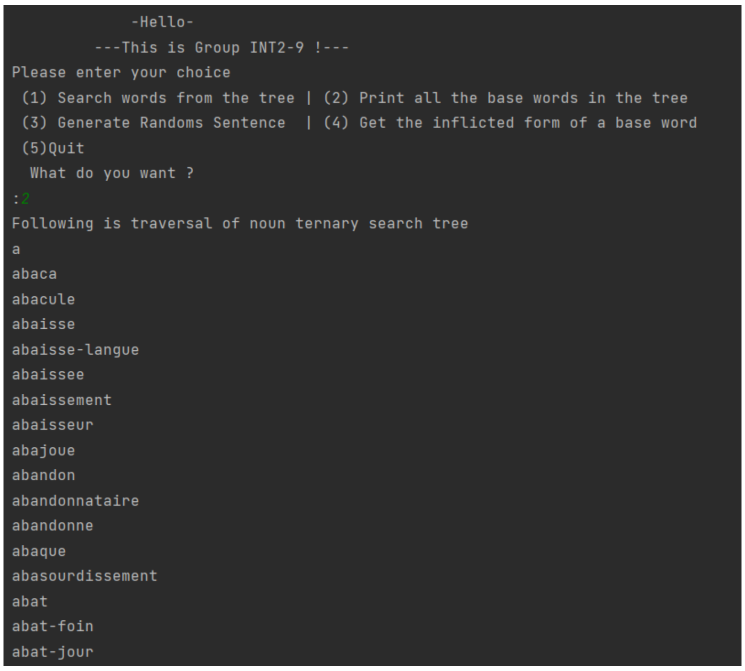
2 - Construct the tree containing the nouns, adjectives, etc.

We chose to divide our work into **one tree for each type of word** : nouns, adjectives, verbs and adverbs. This way, when searching for a noun, we will visit only the tree containing nouns. This **reduces the complexity** when we are searching for a word of a given type.

The link below shows how the words are implemented in the tree :

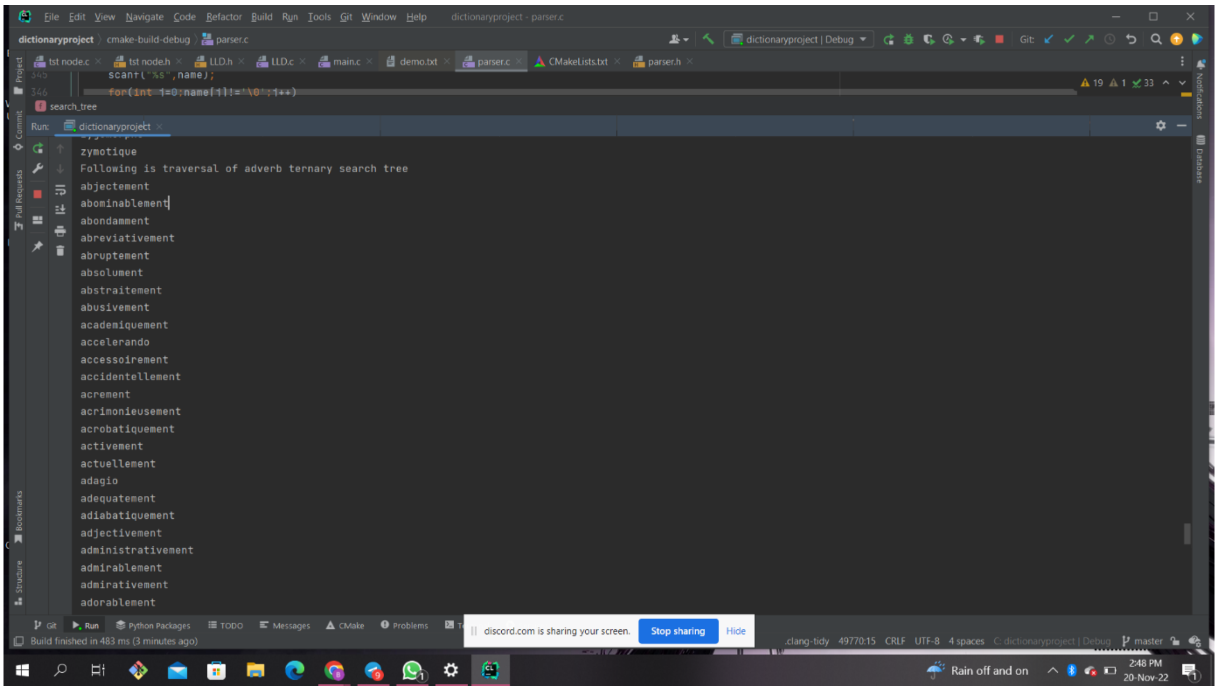
https://www.cs.usfca.edu/~galles/visualization/TST.html

We took screenshots showing the list of all the words of the dictionary. We have one tree for each type. We have nouns, adjectives, verbs and adverbs.



```
              -Hello-
          ---This is Group INT2-9 !---
Please enter your choice
 (1) Search words from the tree | (2) Print all the base words in the tree
 (3) Generate Randoms Sentence  | (4) Get the inflicted form of a base word
 (5)Quit
  What do you want ?
:2
Following is traversal of noun ternary search tree
a
abaca
abacule
abaisse
abaisse-langue
abaissee
abaissement
abaisseur
abajoue
abandon
abandonnataire
abandonne
abaque
abasourdissement
abat
abat-foin
abat-jour
```

Above is the tree containing **nouns** base forms. All the other trees are displayed below those lines. We have to scroll down or search ‘verbs’ ‘adverbs’ or ‘adjectives’ to access the other trees. We’ll show the display of **the ‘adverbs’ below** and there is the same thing for verbs and adjectives too.

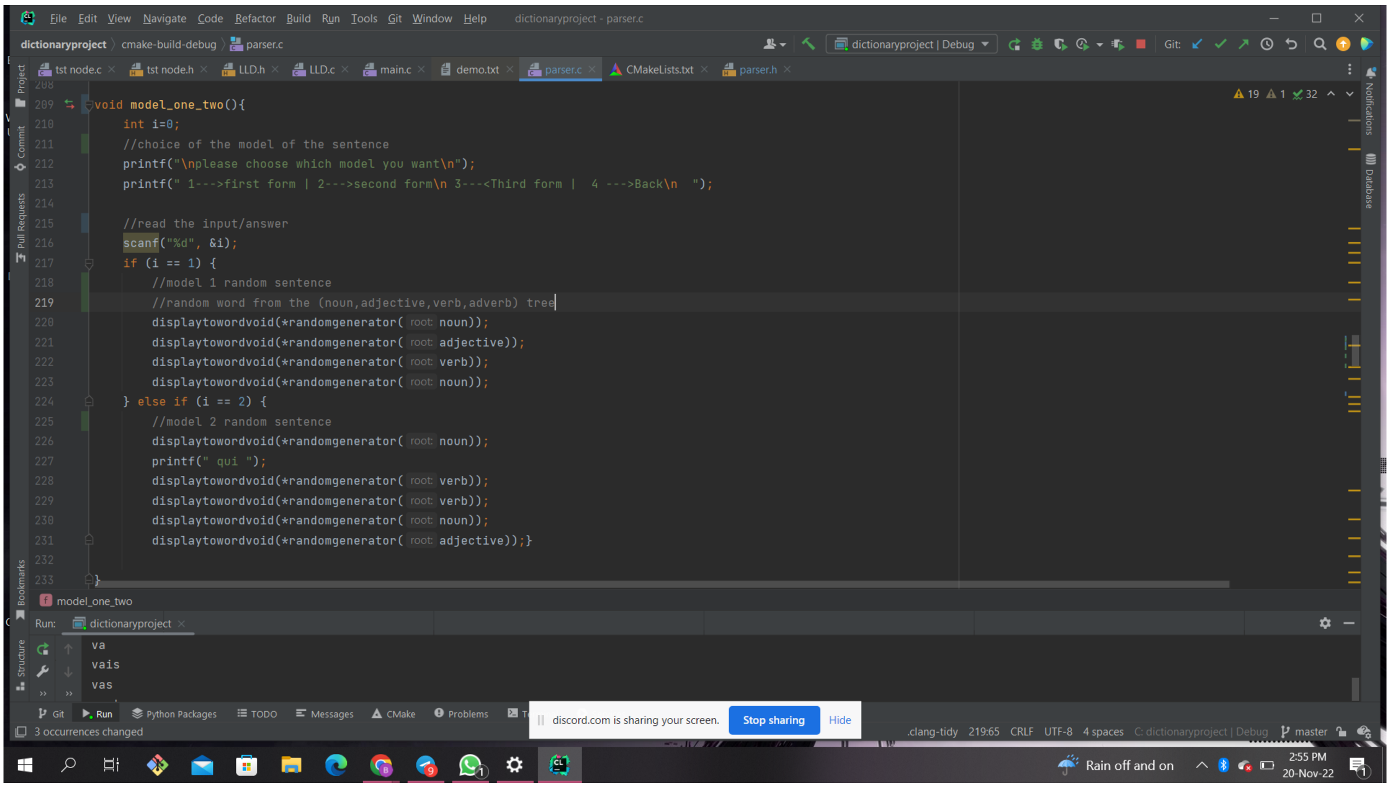

File Edit View Navigate Code Refactor Build Run Tools Git Window Help
dictionaryproject - parser.c
dictionaryproject cmake-build-debug parser.c
tst node.c tst node.h LLD.h LLD.c main.c demo.txt parser.c CMakeLists.txt parser.h
Run: dictionaryproject
Git Run Python Packages TODO Messages CMake Problems
discord.com is sharing your screen. Stop sharing Hide

```
    scanf("%s",name);
    for(int i=0;name[i]!='\0';i++)
```

```
zymotique
Following is traversal of adverb ternary search tree
abjectement
abominablement
abondamment
abreviativement
abruptement
absolument
abstraitement
abusivement
academiquement
accelerando
accessoirement
accidentellement
acrement
acrimonieusement
acrobatiquement
activement
actuellement
adagio
adequatement
adiabatiquement
adjectivement
administrativement
admirablement
admirativement
adorablement
```

Build finished in 483 ms (3 minutes ago)

Then in the screenshot below, we can see the function that generates and display **random sentences** from model 1 and 2.



```c
void model_one_two(){
    int i=0;
    //choice of the model of the sentence
    printf("\nplease choose which model you want\n");
    printf(" 1--->first form | 2--->second form\n 3---<Third form |  4 --->Back\n  ");

    //read the input/answer
    scanf("%d", &i);
    if (i == 1) {
        //model 1 random sentence
        //random word from the (noun,adjective,verb,adverb) tree
        displaytowordvoid(*randomgenerator( root: noun));
        displaytowordvoid(*randomgenerator( root: adjective));
        displaytowordvoid(*randomgenerator( root: verb));
        displaytowordvoid(*randomgenerator( root: noun));
    } else if (i == 2) {
        //model 2 random sentence
        displaytowordvoid(*randomgenerator( root: noun));
        printf(" qui ");
        displaytowordvoid(*randomgenerator( root: verb));
        displaytowordvoid(*randomgenerator( root: verb));
        displaytowordvoid(*randomgenerator( root: noun));
        displaytowordvoid(*randomgenerator( root: adjective));}

}
```

```
va
vais
vas
```

3 occurrences changed

discord.com is sharing your screen.

3 - Word research among base forms :

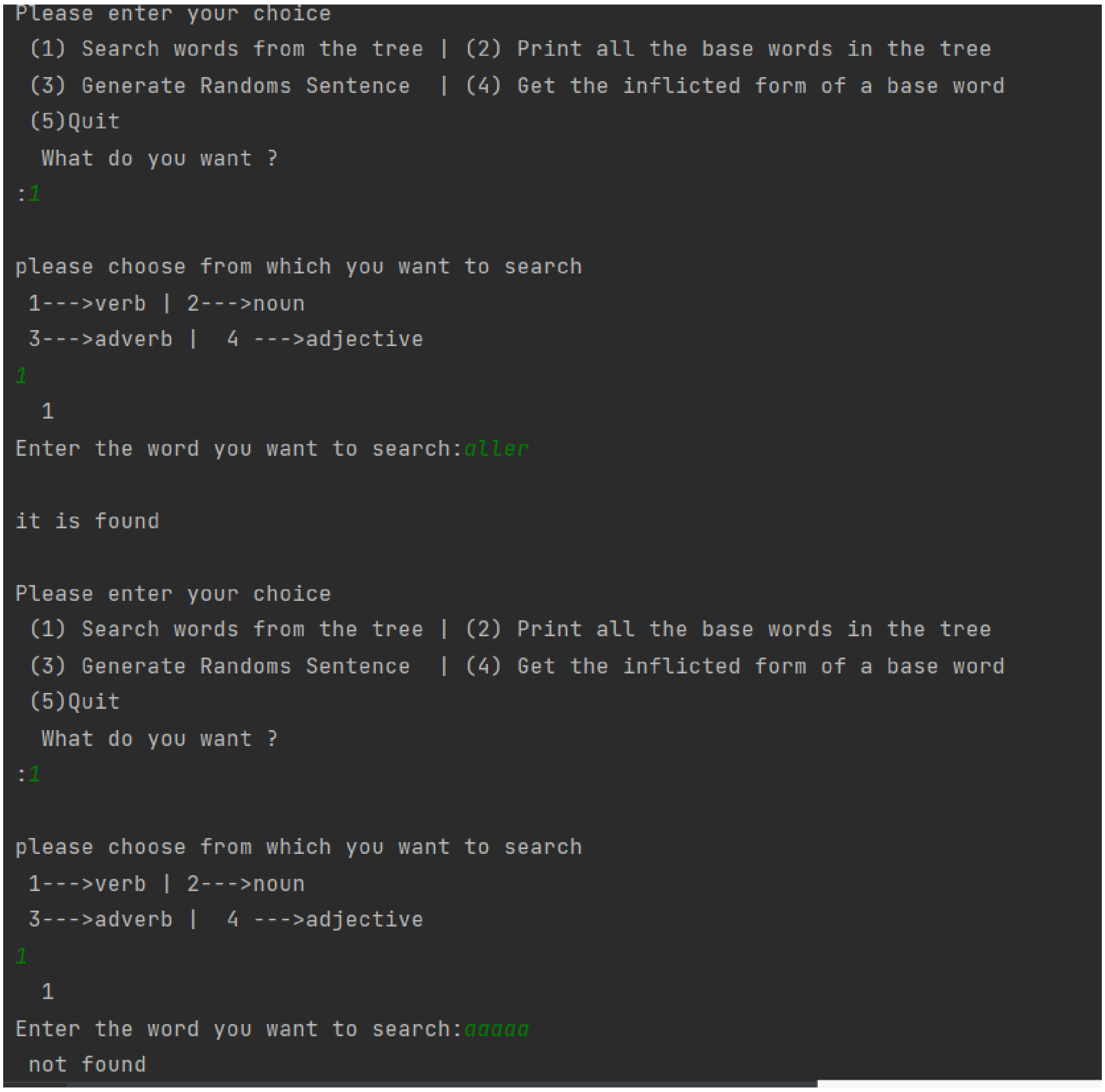
This function will indicate whether or not the word researched has been found in our tree. When doing the tree visit in a ternary tree, we can go through **3 pathes : either left, right or equ**al (which corresponds to below on a scheme). We create the variable temp to visit the tree, and for each node, we compare temp with the letter we are looking for. There are 3 possible outcomes to this operation :

1. Temp->value == letter;

   Then, temp takes temp->equal

2. Temp->value<letter;
   Then, temp takes temp-> left
3. Temp->value>letter;
   Then temp takes temp->right.

Then, this function will repeat this process until it finds the base form that we are searching for. If the value we are searching for is stored in the tree, it will return **'found'**, and otherwise 'not found'.


```
Please enter your choice
 (1) Search words from the tree | (2) Print all the base words in the tree
 (3) Generate Randoms Sentence  | (4) Get the inflicted form of a base word
 (5)Quit
  What do you want ?
:1

please choose from which you want to search
 1--->verb | 2--->noun
 3--->adverb |  4 --->adjective
1
  1
Enter the word you want to search:aller

it is found

Please enter your choice
 (1) Search words from the tree | (2) Print all the base words in the tree
 (3) Generate Randoms Sentence  | (4) Get the inflicted form of a base word
 (5)Quit
  What do you want ?
:1

please choose from which you want to search
 1--->verb | 2--->noun
 3--->adverb |  4 --->adjective
1
  1
Enter the word you want to search:aaaaa
 not found
```

In the screenshot above, we decided to take two examples : **a word that exist : 'aller'**, and our **program return 'found'**. Then another word that **doesn't exist 'aaaaa'** and the **program return 'not found**'. This shows that our function can recognize the words that are in the dictionary

4 - Extract a random base form

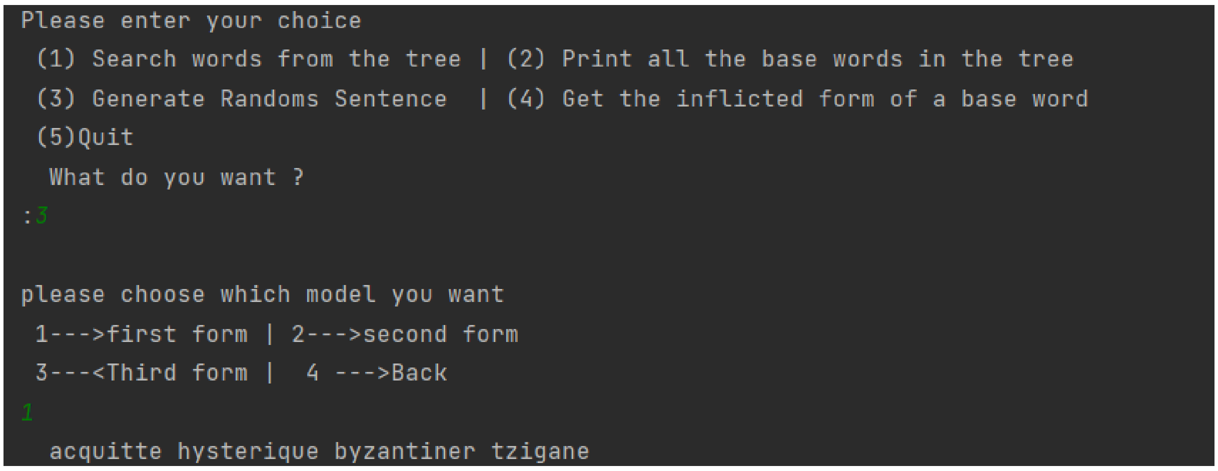
The random word will be stored in a list. We visit the tree using a **random function** between 1 and 3. The outcome of this operation will determine the path we will follow : left, equal or right. When going through **temp->equal**, we will retain the value of temp by **adding it to the list**. When moving right or left, the previous letter is not retained. Also, since we have to end the word at some point, the function will test if the node if leaf for each node. And if the node is leaf, it will either return the word we have stored in the list, or not. In the case where the node has no son, it will return the word. Otherwise, it depends on a random function that will determine if we want to go further in the tree. This function is made so that we can go far enough in the tree to have long words. Otherwise, the function would return a word whenever we pass through a leaf node. This would return only short words.

5 - Random sentence generator

Explanation of the algorithm : For those functions, we needed to use a **condition**, so as to find the type of word required (for example while node->type != Noun). We use the random base form extraction function until we find the desired type of word (for example noun), then we repeat this to find the adjective, verb and last noun. This algorithm works for the 2 models in base forms. Then, for inflicted forms sentences, we use the algorithm for base forms, but the difference is that we add a function to pick up a random inflicted word, inside the tree pointed by the base form that we found through the first function.

A - Base forms

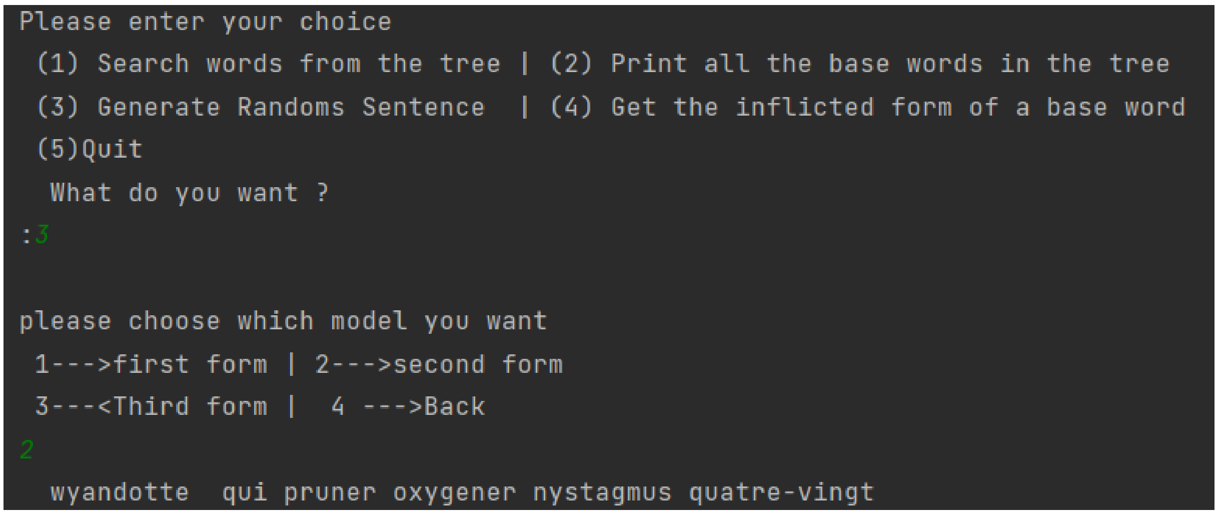
Test of model 1 :



```
Please enter your choice
 (1) Search words from the tree | (2) Print all the base words in the tree
 (3) Generate Randoms Sentence  | (4) Get the inflicted form of a base word
 (5)Quit
  What do you want ?
:3

please choose which model you want
 1--->first form | 2--->second form
 3---<Third form |  4 --->Back
1
  acquitte hysterique byzantiner tzigane
```

The 1st model for base forms works

Test of model 2 :



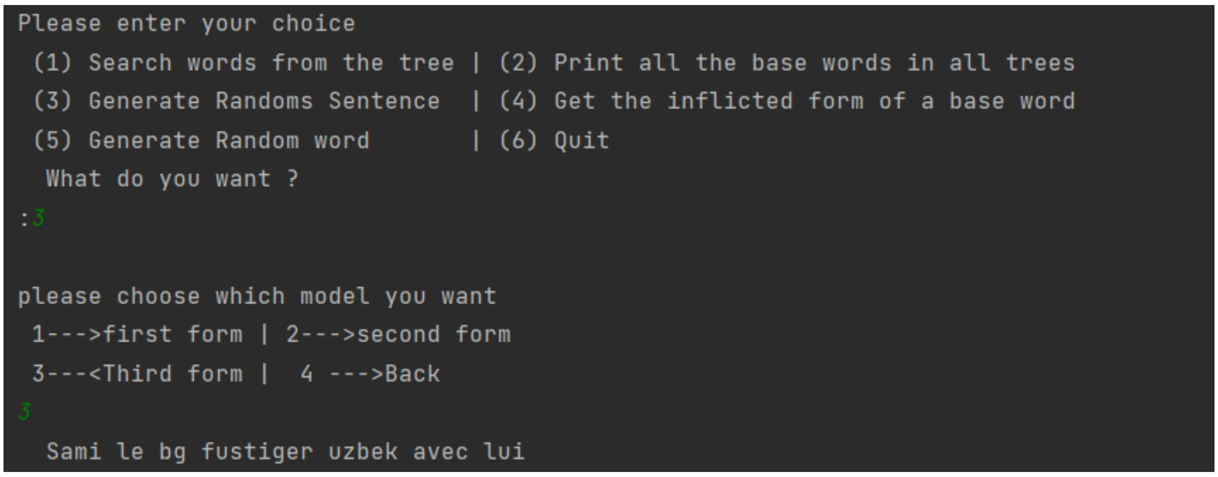
```
Please enter your choice
 (1) Search words from the tree | (2) Print all the base words in the tree
 (3) Generate Randoms Sentence   | (4) Get the inflicted form of a base word
 (5)Quit
  What do you want ?
:3

please choose which model you want
 1--->first form | 2--->second form
 3---<Third form |  4 --->Back
2
  wyandotte  qui pruner oxygener nystagmus quatre-vingt
```

Here we can see that the 2nd model from base forms works. We even discovered new words because we didn't know the existence of "Wyandotte" in French.

```
wyandotte   wyandotte  Adj:InvGen+SG
wyandotte  wyandotte  Nom:Fem+SG
```

Personal model test :

Our personal model is the following : The first word is either Sami Axel or Bamlak, based on a rand function followed by "le bg". Then we chosed to pick a random verb among the base form tree followed by a random noun and random adjective. Finally we put "avec" and "lui/elle" based on a rand function. There is an example below.



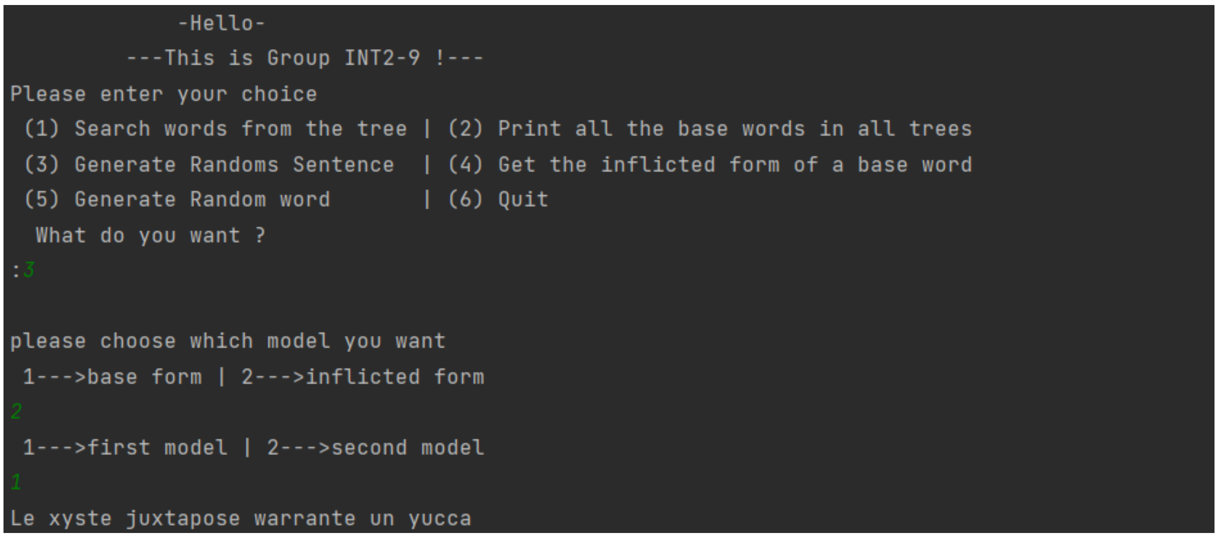
```
Please enter your choice
 (1) Search words from the tree | (2) Print all the base words in all trees
 (3) Generate Randoms Sentence   | (4) Get the inflicted form of a base word
 (5) Generate Random word        | (6) Quit
  What do you want ?
:3

please choose which model you want
 1--->first form | 2--->second form
 3---<Third form |  4 --->Back
3
  Sami le bg fustiger uzbek avec lui
```

B – Inflicted forms

For inflicted forms, we had to make **grammatically correct sentences**. This induces that we had to keep a coherence with the plurality and the genre of the sentence. In our opinion this was quite difficult because we had to access first, the leaf node of the base form, then the leaf node of a random inflicted form : ‘inf_Node’, and then get access to the genre and plurality of the word, which we stored in a structure called ‘inf_Type‘, pointed by ‘inf_Node’.

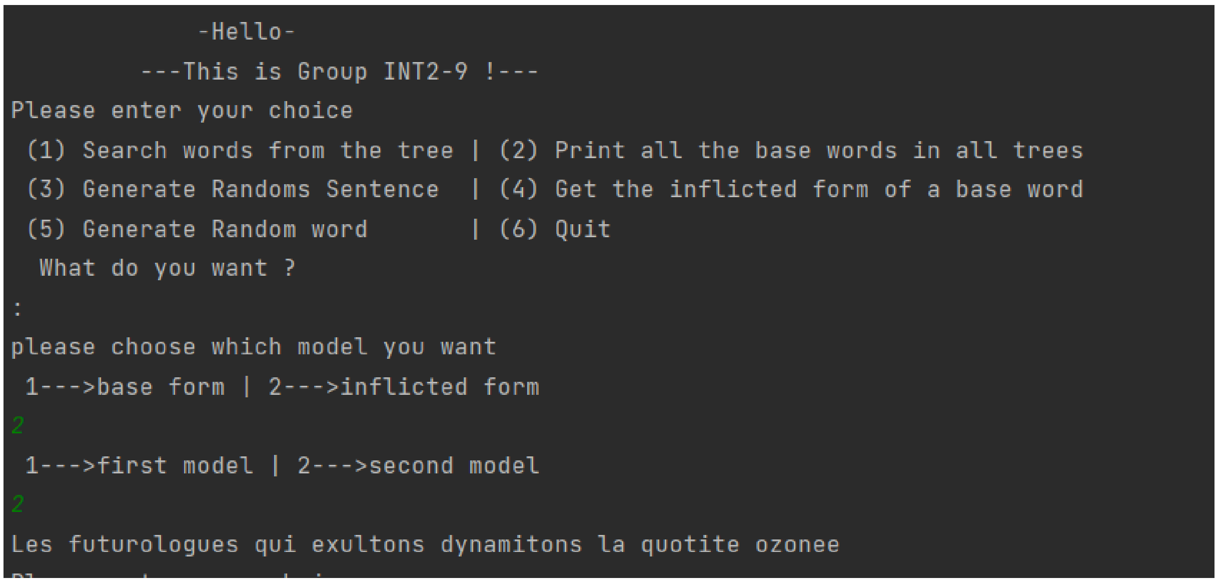
Below are functions that we used to generate random sentences with inflicted forms.

Test of model 1



```
             -Hello-
         ---This is Group INT2-9 !---
Please enter your choice
 (1) Search words from the tree | (2) Print all the base words in all trees
 (3) Generate Randoms Sentence  | (4) Get the inflicted form of a base word
 (5) Generate Random word       | (6) Quit
  What do you want ?
:3

please choose which model you want
 1--->base form | 2--->inflicted form
2
 1--->first model | 2--->second model
1
Le xyste juxtapose warrante un yucca
```

Test of model 2



```
             -Hello-
         ---This is Group INT2-9 !---
Please enter your choice
 (1) Search words from the tree | (2) Print all the base words in all trees
 (3) Generate Randoms Sentence  | (4) Get the inflicted form of a base word
 (5) Generate Random word       | (6) Quit
  What do you want ?
:
please choose which model you want
 1--->base form | 2--->inflicted form
2
 1--->first model | 2--->second model
2
Les futurologues qui exultons dynamitons la quotite ozonee
```

6 – Research an inflicted form :

For this function, the algorithm we chose to implement can be explained through an example. Let's say the user typed 'alle'. Then, the algorithm will search for all the base forms starting by 'alle', for example 'aller' and 'allegorie'. Those words will be stored in a new list.

Then, the algorithm will also store in the list the inflicted forms starting by 'alle' contained in the tree pointed by the base forms starting by 'alle'. The algorithm will repeat the same with 'all', then 'al', then 'a'. This algorithm permits to limit the complexity. Indeed, we will search first in the trees that are the more likely to contain the inflicted form that we are looking for. This is simply explained by the fact that inflicted forms are more likely to be associated with a base form that shares letters in common.

Though this algorithm has 1 default that we are aware of : in exceptions like 'vais' that is an inflicted form of 'avoir', our algorithm will not be able to find 'vais', unless we search randomly in all the trees that we made. We have not been able to solve this case in time. But we are aware that our algorithm works for 99% of cases so we are still satisfied of the result. **Unfortunately we couldn't do this function in time.**

Conclusion :

We are very satisfied about the fact that we managed to make most functions required by the project. It's due to the fact that we managed to implement the words in the tree quite fast, which let us much time to do the functions. Every member of the group feels like we have all contributed to the project. What really helped us is the fact that we had different opinions on how we should solve this project, and thanks to ideas sharing, we managed to find the best solution to code this dictionary, in our opinion. For instance, Axel thought we should do a n-ary tree where each node has a different number sons, and Sami was arguing with Axel at first. Whereas Bamlak wanted to use a ternary tree. Finally, by arguing and trying to use those 2 types of trees, we came to the conclusion that we should use the ternary tree. We understood that having different ideas is actually better to advance in a project, rather than everyone having the same idea. Thanks to this, we tried a greater number of possibilities concerning the choice of the structure for example.

Thanks to all the hours invested on this project, we have learnt many technical stuff in c. Firstly, we became familiar with the ternary trees, which is most likely going to be helpful for our studies. We learnt how to extract text from a .txt file. We also performed many functions to help visiting a tree, which are going to be helpful for this year's course. Moreover, we have done useful functions such as automatic completion that we will be able to reuse for other algorithms.

Finally, we have learnt plenty of communication and organization qualities. Since, we had a short delay for the project, we had a short time to plan our work, and I think we handled this quite well. Also, we managed to use our divergent opinion to make good solutions for the functions . To put in a nutshell, communication and investment where the keys to perform this project.