ICT School-based Assessment
Grandmaly - A Composition Analyzer

Choi Hoi To
6A 06
HKTA Tang Hin Memorial Secondary School

Table of content

Introduction

Background

Children are eagerly learning English in primary school. They need to include some of the wordings they have learned in lessons in their writings. They also have to use a wide range of vocabulary. This program is created for primary school teachers to check some basic statistics of their students' composition, including the number of sentences, paragraphs, words, and find if some keywords are included in their piece of work. Also, a file with a list of unique words is generated for teachers for further analysis. It can facilitate efficiency in marking students' work.

Requirement

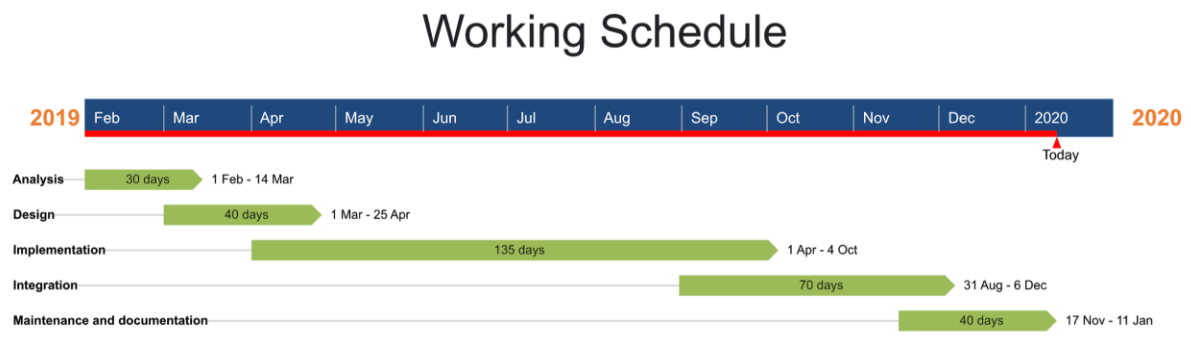
This program should be able to generate basic statistics for the compositions that students have submitted. There also should have a function for teachers to search if some keywords or vocabulary are used in their composition. A list of unique wordings should be generated too.

Solution

To achieve the above requirement, the program is written in Pascal. Some functions and procedures for text files, strings, arrays are used to create this program. A simple GUI is designed in a console environment to create a neat and user-friendly interface for teachers to use the program.

Working Schedule

There are 5 steps in the working schedule, including analysis, design, implementation, integration, and maintenance and documentation.



Design

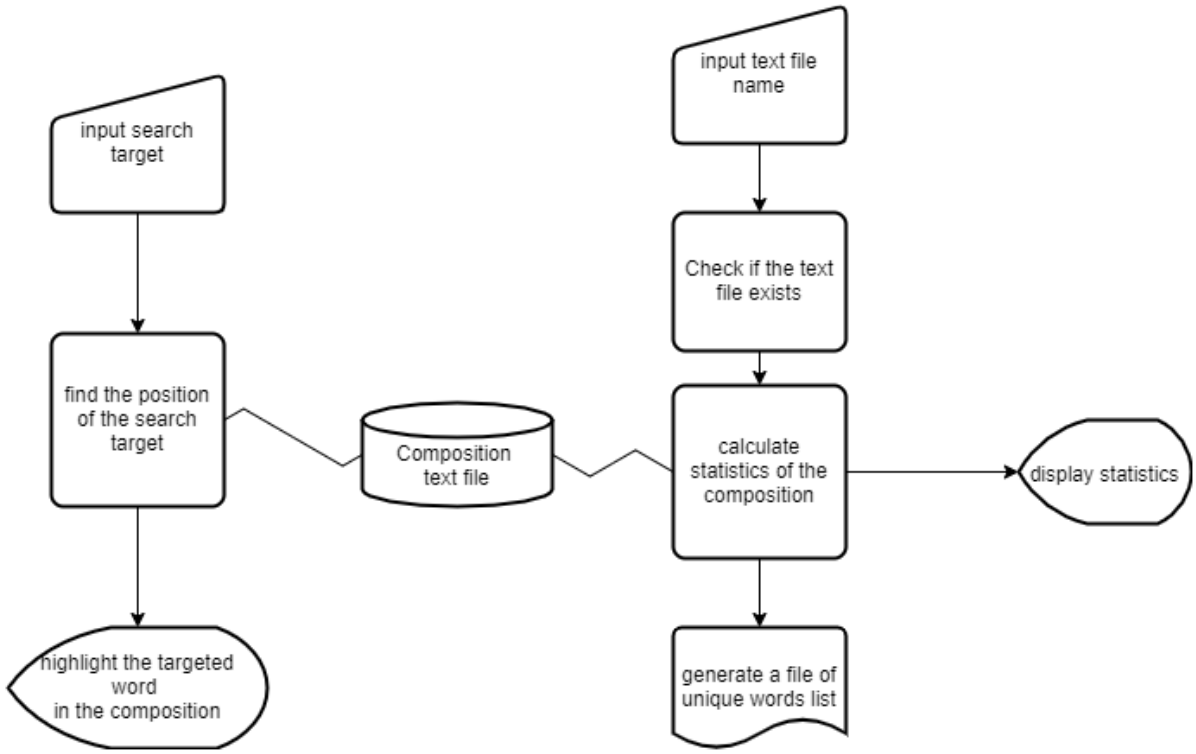
Program structure

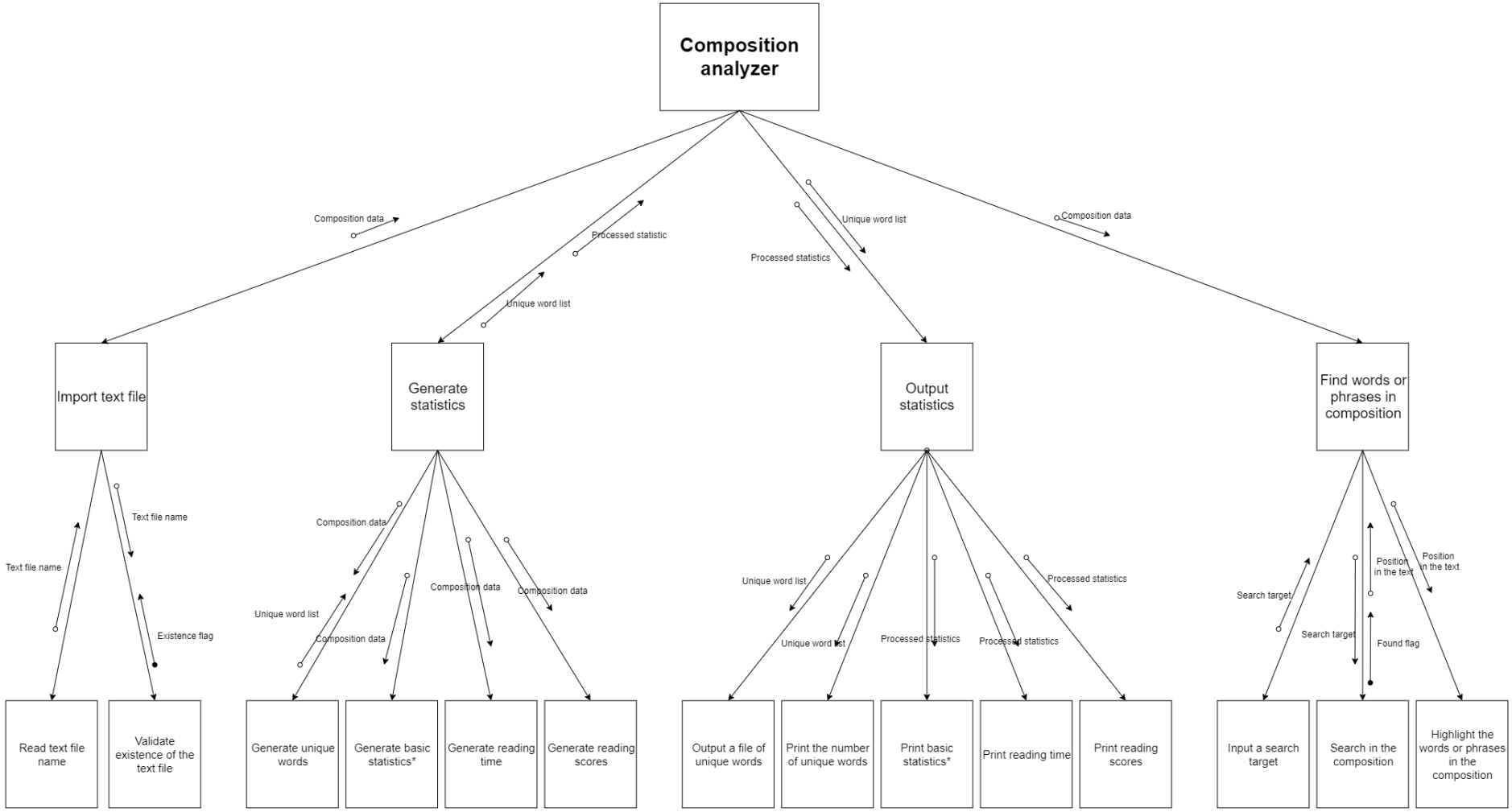
Users can only use a keyboard to control the program. Basically using arrow keys, enter key, escape key and the page up and down key can fully transit to different parts of the program.

The main part of my program is just for determining the next screen. A variable `state` is used to determine which screen that the program is going to move to. Each screen is a procedure, with a variable parameter `state`. When each procedure ends, the global variable `state` will be updated to the number of the next screen. A post-test loop structure is implemented to determine the end of the program. Inside the loop, there is a `case`. It helps in determining the next screen. If `state` equals -1, the program can escape the loop and ends the program.

The whole program is separated into different units, including `passageAnalyser`, `longStringType`, `stringListType`, and `ui`. `passageAnalyser` is a unit storing all functions for passage-analyzing, `longStringType` is a unit creating a custom type `longString` and storing its functions, `stringListType` is a unit creating a custom type `stringList` and storing its functions, and `ui` is a unit storing all procedures of drawing different UI components. The modularity of units facilitates code reuse.

Algorithm design




Composition analyzer
Composition data
Processed statistic
Unique word list
Processed statistics
Unique word list
Composition data
Import text file
Generate statistics
Output statistics
Find words or phrases in composition
Text file name
Text file name
Existence flag
Composition data
Unique word list
Composition data
Composition data
Composition data
Unique word list
Processed statistics
Unique word list
Processed statistics
Processed statistics
Search target
Position in the text
Position in the text
Search target
Found flag
Read text file name
Validate existence of the text file
Generate unique words
Generate basic statistics*
Generate reading time
Generate reading scores
Output a file of unique words
Print the number of unique words
Print basic statistics*
Print reading time
Print reading scores
Input a search target
Search in the composition
Highlight the words or phrases in the composition

In Pascal, the `string` type is an array of characters, with only 255 characters as the maximum length. 255 characters are not enough to store the whole piece of composition, so a new type `longString` is created. `longString` is a 0-based dynamic array of characters. Each character in the text file is stored into `longString`. It works similar to the original `string`. The differences are that `string` is 1-based and `longString` is 0-based, and some of the string functions have to be rewritten.

This program generates basic statistics of the composition, like the number of words, paragraphs, sentences, characters, etc. This information can be analyzed by manipulating the long string storing the composition. A list of unique words is also generated by adding unique words to `stringList`, an array of strings. When a new word is extracted from the composition, the program will check if that word is already in the list, then add the word to the list if not. This can make sure a list of unique words can be generated.

There is also a find function for searching words or phrases in the composition. When a user typed a searching target, the position of the target in the `longString` of the composition will be returned. After tones of calculation, the corresponding position on the screen will be highlighted.

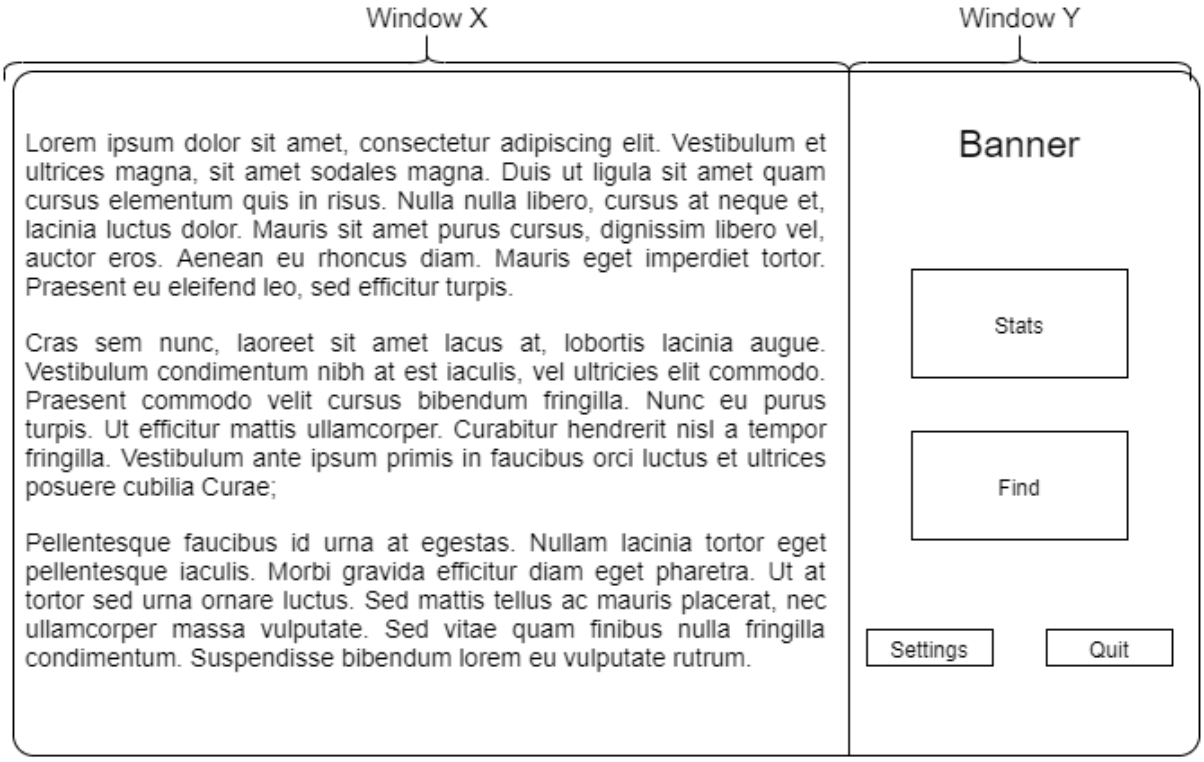
UI design

To achieve a GUI like experience in a console environment, a unit `ui` is written. It provides different kinds of UI components that we usually see in a GUI, namely message box, input box, button, checkbox. All the above-mentioned components in this unit are solely printed out in the console. All their behaviors are being controlled in the main program for convenience.

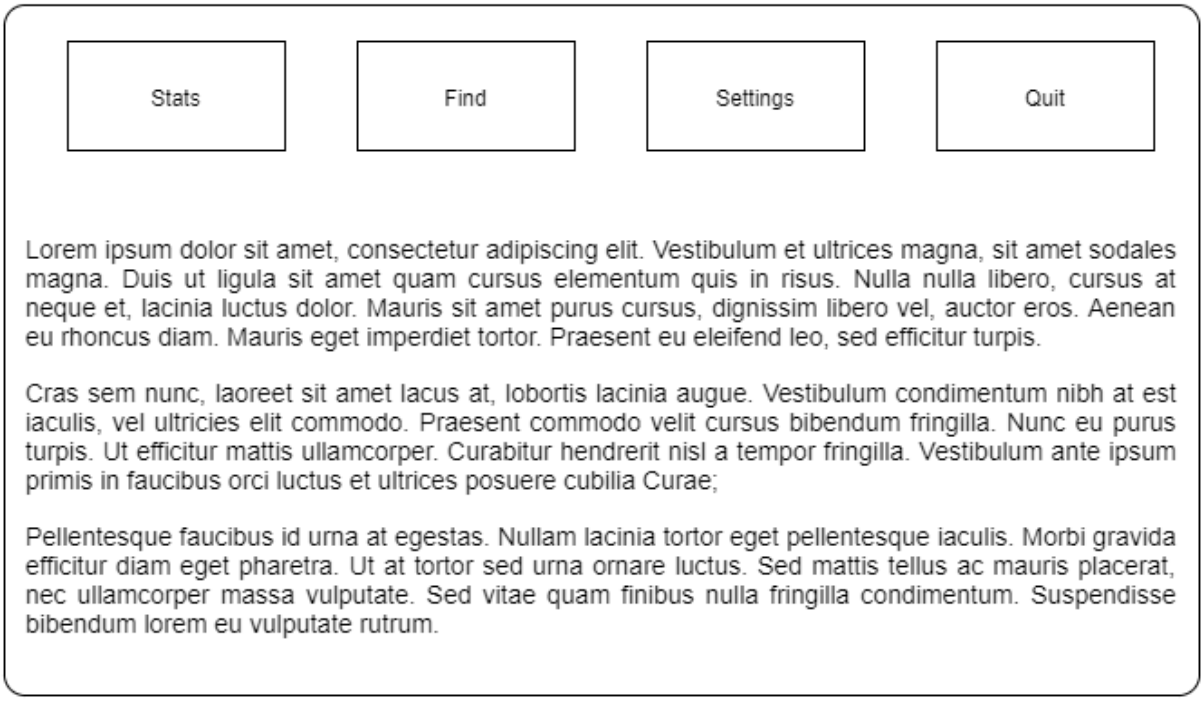
This program contains 6 screens, including:

- checkScreenWidthScreen
  - It checks if the width of the console matches the dimension in the program(i.e. Width 120).
- importTextFileScreen (`state` = 0)
  - It allows users to type the path of their composition to be analyzed.
- mainScreen (`state` = 1)
  - It allows users to choose the functions they want to use through up and down arrow keys to select different buttons.
- statsScreen (`state` = 2)
  - Statistics of the composition are shown on this screen.
- findScreen (`state` = 3)
  - Specific words or phrases can be searched in the composition. If they are found, those words will be highlighted.
- settingsScreen (`state` = 4)
  - It shows the current composition that the program is analyzing, and allows users to return to importTextFileScreen to analyze another composition.

This is the drafted UI of the program:



The UI is designed in this way for a clean and user-friendly user experience. A large portion of the screen (Window X) is set to display the passage, while Window Y is for users to choose the functions he/she wants to use. When moving on to other screens, only the contents in Window Y will be updated and keeping the whole passage showing in Window X. For example, in statsScreen, statistics of the composition will be shown on Window Y, and in findScreen, a search box will be shown on Window Y, while Window X displays the whole composition. This can keep the UI of the program consistent, and will not confuse users easily if the control parts on the UI are changing all the time.



This is another draft of my UI design. But in the above UI design, the composition is displayed too widely. Users can follow and read the composition more easily if the width of its display area is smaller. Also, if the buttons are pressed, the composition displayed will either be erased or moved to another position, causing inconvenience to the user. Users cannot compare the statistics and the original text of the passages side by side. So this design is not preferred.

Implementation

Custom long string type

```
type longString = array of Char;
```

Count the occurrence of a specific character in the long string

```
function countInLongString(c : Char; s :
longString) : Integer;
var i : Integer;
begin
    countInLongString := 0;
    for i := 0 to Length(S)-1 do
        if lowerCase(s[i]) = lowerCase(c) then
            countInLongString :=
countInLongString + 1;
end;
```

If `s[i]` matches `c`, then the value is increased by 1.

Get the position of the first occurrence of a specific characters

```
function posOfChar(c : char; s : longString) :
Integer;
var i : Integer;
begin
    i := 0;
    posOfChar := -1;
    while (i <= Length(s)-1) and (s[i] <> c)  do
        i := i + 1;
    if i < Length(s) then
        posOfChar := i;
end;
```

If `s[i]` is not equal to `c`, the pointer of the long string `i` is increased by 1.

Generate an array of positions of the substring in the long string

```
type integerArray = array of Integer;
function posOfString(sub: string; long:
longString): integerArray;
var i, j, n, temp : Integer;
begin
    posOfString := Nil;
    n := 0;
    for i := 0 to Length(long)-Length(sub) do
    begin
        j := 1;
        temp := i;
        while (j <= Length(sub)) and (sub[j] =
long[temp]) do
        begin
            j := j + 1;
            temp := temp + 1;
        end;
        if j-1 = Length(sub) then
        begin
            n := n + 1;
            SetLength(posOfString, n);
            posOfString[n-1] := i;
        end;
    end;
end;
```

Initialization of the array.

`i` is the pointer of the long string.

`j` is the pointer of the substring.

If the first character of the substring matches the current character of the long string, `j` and `i` are increased by 1 to check the next character.

If the substring is a part of the long string, then add `i` to the integer array to store its position.

Converting long string into an array of string within a fixed length

```
type passageArray = array of string;
procedure splitLongStringToArray(s : longString; var
a : passageArray);
var i, x, n : Integer;
    temp : string;
begin
    i := 0;
    temp := '';
    n := 1;
    SetLength(a, n);
    a[n-1] := '';
    while i < Length(s) do
    begin
        if (s[i] in words_set) then
            temp := temp + s[i]
        else if not(s[i] in [#13, #10]) then
        begin
            x := Length(a[n-1]) - 1;
            if x + Length(temp) >= screen_width then
            begin
                n := n + 1;
                SetLength(a, n);
                a[n-1] := '';
            end;
            a[n-1] := a[n-1] + temp + s[i];
            temp := '';
        end
        else if (s[i] = #13) and (s[i+1] = #10) then
        begin
            n := n + 2;
            SetLength(a, n);
            a[n-2] := '  ';
        end;
        i := i + 1;
    end;
end;
```

This procedure is used for text wrapping when displaying the composition.

`i` is the pointer of the long string.
`temp` is used to store the word extracted.
`n` is the size of the `passageArray`.

Add `s[i]` to `temp` if it is a valid alphabet.

If `s[i]` is a space or a punctuation mark, it will check if the length of the string plus the length of `temp` exceeds the screen width. If the screen width is exceeded, `temp` will be append to the next element of the `passageArray`.

`temp` is reset after appending to the `passageArray`.
If `s[i]` is a line break, then add two null elements to the `passageArray`.

Custom string list type

```
type stringList = array of string;
```

Adding new entry to the string list

```
procedure add(var l : stringList; s : string);
var i : Integer;
begin
    SetLength(l, Length(l) + 1);
    l[Length(l)-1] := s;
    i := Length(l)-1;
    while (l[i-1] > l[i]) and (i > 0) do
    begin
        swap(l[i-1], l[i]);
        i := i - 1;
    end;
end;
```

Increase the size of the array by 1. The string is added to the end of the array.

Sort the array of strings in ascending order.

Checking the presence of a string in the string list

```
function contains(l : stringList; s : string):
Boolean;
begin
    contains := binarySearch(l, s, 0, Length(l)-
1);
end;
```

The binarySearch function returns a boolean to check if the string is present in the string list.

Binary search for searching the string list

```
function binarySearch(a : stringList; data :
string; first, last : Integer): Boolean;
var mid : Integer;
begin
    mid := (first + last) div 2;
    if (last >= first) then
        if a[mid] = data then
            binarySearch := true
        else if a[mid] > data then
            binarySearch := binarySearch(a,
data, first, mid - 1)
        else
            binarySearch := binarySearch(a,
data, mid + 1, last)
    else
        binarySearch := false;
end;
```

`mid` is the pointer of the middle element in the array.

If the middle element matches the search target, `true` is returned.

Returning the size of the string list

```
function size(l : stringList): Integer;
begin
    size := Length(l);
end;
```

The length of the array is returned.

Composition analyzing

Counting the number of characters

```
function countNoOfChar(s : longString):
Integer;
begin
    countNoOfChar := Length(s) -
countInLongString(#10, s) -
countInLongString(#13, s);
end;
```

The number of characters is calculated by deducting the total length of the composition by the number of characters with ASCII values 13 and 10 in it, assuming that there are only these two invisible characters in the long string.

Counting the number of words

```
function countNoOfWords(s : longString):
Integer;
begin
    countNoOfWords := countInLongString('
', s) + countInLongString(#13 + #10, s) +
1;
end;
```

The number of words is calculated by counting the total number of spaces and line breaks in the composition, and finally adding 1 to the value.

Counting the number of paragraphs

```
function countNoOfPara(s : longString):
Integer;
begin
    countNoOfPara :=
countInLongString(#13 + #10, s) + 1;
end;
```

The number of paragraphs is calculated by counting the number of line breaks in the composition, and finally adding 1 to the value.

Counting the number of sentences

```
function countNoOfSentences(s :
longString): Integer;
const eos_marks = ['.', '?', '!'];
      words_set = ['A'..'Z', 'a'..'z',
'0'..'9'];
var i : Integer;
begin
    countNoOfSentences := 0;
    for i := 0 to Length(s)-1 do
        if (s[i] in eos_marks) and ((i =
Length(s) - 1) or not(s[i+1] in words_set))
then
            countNoOfSentences :=
countNoOfSentences + 1
end;
```

These punctuation marks represents the end of a sentence. These characters represent the valid characters in a word.

If `s[i]` matches the valid punctuation marks above, and the `s[i+1]` is not an alphabet, then `s[i]` represents the end of the sentence. Then the number of sentences increases by 1.

Calculating reading ease score

```
function generateReadingEaseScore(s :
longString; noOfWords, noOfSent :
Integer):Real;
var i : Integer;
begin
    i := countInLongString('a', s) +
countInLongString('e', s)
+countInLongString('i', s) +
countInLongString('o', s) +
countInLongString('u', s);
    generateReadingEaseScore := 206.835 -
(1.015 * (noOfWords/noOfSent)) - (84.6 *
(i/noOfWords));
end;
```

A reading ease score is calculated using the Flesch reading-ease score formula[1]. The total number of vowels has to be counted first to calculate the reading ease score.

[1] Source: https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests

Calculating the reading time

```
function generateReadingTime(noOfWords :
Integer):string;
begin
    generateReadingTime :=
toMinute(noOfWords / 200);
end;
```

The reading time in words per minute is generated by dividing the number of words by 200[2]. Then the display time will be generated by a function `toMinute()`.

Returning the string of reading time

```
function toMinute(r : Real): string;
var m, s : string;
begin
    Str(trunc(r), m);
    Str(frac(r)*60:0:0, s);
    toMinute := m + ':' + s;
end;
```

Extracting the integer part of the reading time to minute, and then convert the decimal part of the reading time to seconds. Finally, combine them into a string and return it.

[2] Source: https://ezinearticles.com/?What-is-the-Average-Reading-Speed-and-the-Best-Rate-of-Reading?&id=2298503

Generate a list of unique words

```
function getListOfUniqueWords(s : longString):
stringList;
var i : Integer;
    temp : string;
begin
    clear(getListOfUniqueWords);
    temp := '';
    for i := 0 to Length(s)-1 do
        if (s[i] in words_set) then
            temp := temp + lowerCase(s[i])
        else if (i <> Length(s)-1) and ((s[i] in
['.',''']) and (s[i-1] in words_set) and
(s[i+1] in words_set)) then
            temp := temp + lowerCase(s[i])
        else
        begin
            if (temp <> '') and
((size(getListOfUniqueWords) = 0) or
not(contains(getListOfUniqueWords, temp))) then
                add(getListOfUniqueWords, temp);
            temp := '';
        end;
end;
```

Initialization of the unique word list. `temp` stores the words extracted.

Check if the character (`s[i]`) is an alphabet. If yes then add it to the `temp`.
If `s[i]` is a punctuation mark . or ‘, and the previous character is an alphabet, add `s[i]` into `temp`.

If `s[i]` is neither an alphabet nor a punctuation in a word, then add `temp` into the unique word list if `temp` has not been added to the list before.

Generate a file of unique words

```
procedure generateUniqueWordsTxtFile(list :
stringList);
const unique_words_txt_file_path =
'unique_words.csv';
var t : text;
    i : Integer;
    temp : Char;
begin
    Assign(t, unique_words_txt_file_path);
    Rewrite(t);
    temp := list[0][1];
    for i := 0 to size(list)-1 do
    begin
        if temp <> list[i][1] then
        begin
            WriteLn(t);
            temp := list[i][1];
        end;
        Write(t, list[i] +',');
    end;
    Close(t);
end;
```

The file generated is a CSV.

`temp` stores the alphabet of the first characters of the word. It is used to determine the new line in the text file.

New line is added to the text file to write the unique words of another starting alphabet.

When looping the unique word list, the program keeps adding the words in the list to a text file.

Words searching in the composition

Generating a set of position of searching target in the long string

```
var positionInLongString : integerArray;
positionInLongString := posOfString(target,
passage);
```

An array of integer is returned.

Processing a set of position in passageArray and starting position

```
type x = record
                positionInPassageArray: Integer;
                start : Integer;
         end;
var p : array of x;
...

    SetLength(p, Length(positionInLongString));
    for i := 0 to Length(p)-1 do
    begin
        m := 0;
        temp := positionInLongString[i];
        while (temp >= 0) and (m < Length(a)) do
        begin
            temp := temp - Length(a[m]);
            m := m + 1;
        end;
        if m <> 0 then
        begin
            temp := temp + Length(a[m-1]);
            p[i].positionInPassageArray := m-1;
        end;
        p[i].start := temp;
    end;
```

`passageArray` is an array of string, storing the split passage.
`positionInPassageArray` is the position of the targeted word in the `passageArray`.
`start` is the start position of the targeted word in the corresponding `passageArray`.

The `p` processed is used for highlighting the search target in the composition.

Create an array of record with the same number of elements as `positionInLongString`.

Calculate the corresponding position to be highlighted in the `passageArray`.

User interface

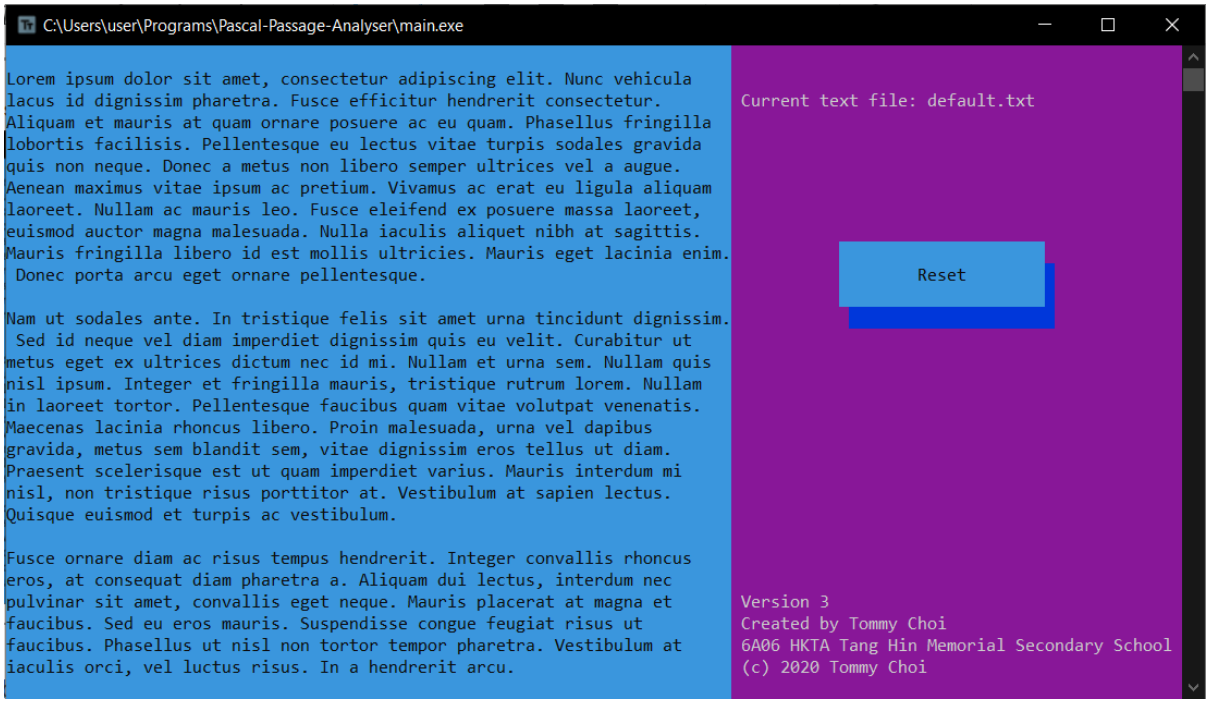
This is the file choosing screen:

This is the main screen of my program:

This is the statistics screen:

This is the find screen:

This is the settings screen:



Custom buttons

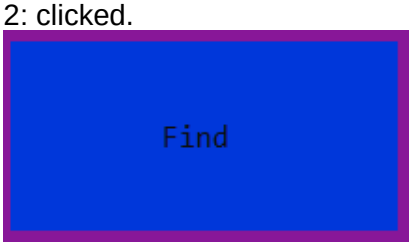
```
procedure drawButton(startX, startY, width, height :
integer; s : string; state : Integer);
var i, j, x, y : integer;
begin
    HighVideo;
    x := width div 2 - Length(s) div 2;
    y := height div 2;
    GotoXY(startX, startY);
    if height = 1 then
    begin
        case state of
            0 : setColors(button_textbackground_normal,
button_textcolor_normal);
            1 : setColors(button_textbackground_selected,
button_textcolor_selected);
            2 : setColors(button_textbackground_clicked,
button_textcolor_clicked);
        end;
        for i := 1 to width do
            Write(' ');
        GotoXY(startX + x, startY);
        Write(s);
    end
    else
        case state of
            0 : begin
                setColors(button_textbackground_normal,
button_textcolor_normal);
                for i := 1 to height do
                begin
                    GotoXY(startX, startY+i-1);
                    for j := 1 to width do
                        Write(' ');
                end;
                GotoXY(startX + x, startY + y);
                Write(s);

setColors(button_textbackground_normal_shadow,
button_textcolor_normal);
                for i := startY + 1 to startY + height do
```

There are 3 states in a button:

0: normal

1: selected

2: clicked.



No shadow will be shown if the height of the button is 1.

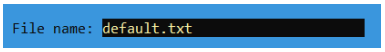
```
                begin
                    GotoXY(startX + width, i);
                    Write(' ');
                end;
                GotoXY(startX + 1, startY + height);
                for i := 1 to width - 1 do
                    Write(' ');
            end;
            1 : begin
                setColors(button_textbackground_selected,
button_textcolor_selected);
                for i := 1 to height do
                begin
                    GotoXY(startX, startY+i-1);
                    for j := 1 to width do
                        Write(' ');
                end;
                GotoXY(startX + x, startY + y);
                Write(s);

setColors(button_textbackground_selected_shadow,
button_textcolor_selected);
                for i := startY + 1 to startY + height do
                begin
                    GotoXY(startX + width, i);
                    Write(' ');
                end;
                GotoXY(startX + 1, startY + height);
                for i := 1 to width - 1 do
                    Write(' ');
            end;
            2 : begin
                setColors(button_textbackground_clicked,
button_textcolor_clicked);
                for i := 1 to height do
                begin
                    GotoXY(startX+1, startY+i);
                    for j := 1 to width do
                        Write(' ');
                end;
                GotoXY(startX + x + 1, startY + y + 1);
                Write(s);
```

```
                setColors(custom_textbackground,
custom_textcolor);
                for i := startY to startY + height -1 do
                begin
                    GotoXY(startX, i);
                    Write(' ');
                end;
                GotoXY(startX + 1, startY);
                for i := 1 to width - 1 do
                    Write(' ');
            end;
        end;
    resetDefaultColors(True, True);
end;
```

Custom input box

```
procedure drawInputBox(startX, startY: Integer;
boxWidth: Integer; message, hint: string; typing :
Boolean);
var i, j : Integer;
begin
    cursoroff;
    setColors(inputbox_textbackground,
inputbox_textcolor);
    if typing then hint := '';
    for i := 1 to 3 do
    begin
        GotoXY(startX, startY + i - 1);
        for j := 1 to 4 + Length(message) + boxWidth do
            Write(' ');
    end;
    GotoXY(startX+1, startY+1);
    Write(message + ': ');
    setColors(inputbox_textbackground_box,
inputbox_textcolor_hint);
    Write(hint);
    for i := 1 to boxWidth - Length(hint) do
        Write(' ');
    resetDefaultColors(True, True);
    cursoron;
end;
```

With hint:

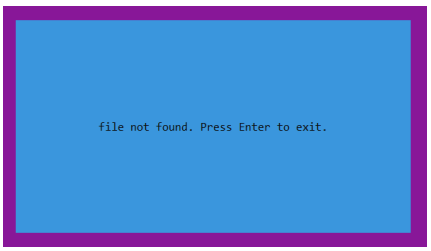


Without hint:

Custom message box

```
procedure drawMsgBox(startX, startY, width, height :
integer; message : string; messageY : Integer);
var i, j, x, y : Integer;
begin
    cursoroff;
    setColors(msgbox_textbackground,
msgbox_textcolor);
    for i := 1 to height do
    begin
        GotoXY(startX, startY + i - 1);
        for j := 1 to width do
            Write(' ');
    end;
    x := width div 2 - Length(message) div 2;
    if (messageY = -1) or (messageY > height-1) then
y := height div 2
    else y := messageY;
    GotoXY(startX + x, startY + y);
    Write(message);
    resetDefaultColors(True, True);
end;
```

`messageY` is the y-position that the `message` will be printed out at. `messageY` should be greater than 0 but cannot exceed `height`. If `messageY` is set to -1, `message` will be printed out on the center of the height of the message box.

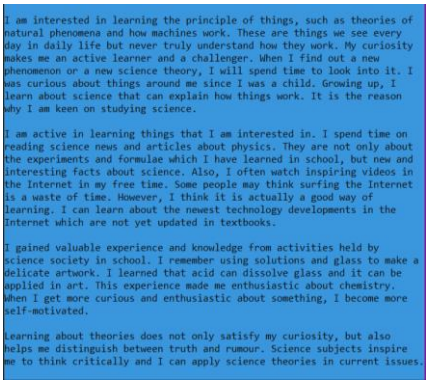


Scrolling of the composition

```
repeat
    c := ReadKey;
    if c = #0 then
    begin
        c := ReadKey;
if (c = #81) and (Length(a) > window1_endY-
window1_startY+1) and (front + window1_endY-
window1_startY < Length(a)-1) then
        begin
            Window(window1_startX, window1_startY,
window1_endX, window1_endY);
            GotoXY(1, 1);
            DelLine;
            front := front + 1;
            GotoXY(1, window1_endY-1);
            Write(a[front + 27]);
        end
        else if (c = #73) and (Length(a) >
window1_endY-window1_startY+1) and (front > 0) then
        begin
            Window(window1_startX, window1_startY,
window1_endX, window1_endY);
            GotoXY(1, 1);
            InsLine;
            front := front - 1;
            Write(a[front]);
        end;
    end;
until c = #13;
```

If the page up key is pressed, the passage will be scrolled up by 1 line.

If the page down key is pressed, the passage will be scrolled down by 1 line.



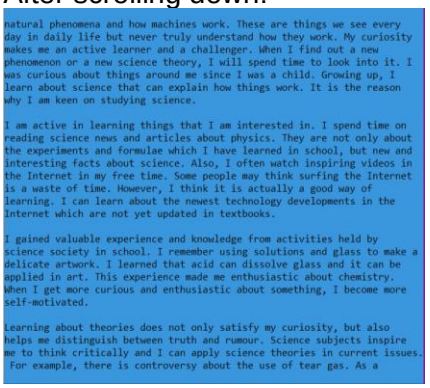
I am interested in learning the principle of things, such as theories of natural phenomena and how machines work. These are things we see every day in daily life but never truly understand how they work. My curiosity makes me an active learner and a challenger. When I find out a new phenomenon or a new science theory, I will spend time to look into it. I was curious about things around me since I was a child. Growing up, I learn about science that can explain how things work. It is the reason why I am keen on studying science.

I am active in learning things that I am interested in. I spend time on reading science news and articles about physics. They are not only about the experiments and formulae which I have learned in school, but new and interesting facts about science. Also, I often watch inspiring videos in the Internet in my free time. Some people may think surfing the Internet is a waste of time. However, I think it is actually a good way of learning. I can learn about the newest technology developments in the Internet which are not yet updated in textbooks.

I gained valuable experience and knowledge from activities held by science society in school. I remember using solutions and glass to make a delicate artwork. I learned that acid can dissolve glass and it can be applied in art. This experience made me enthusiastic about chemistry. When I get more curious and enthusiastic about something, I become more self-motivated.

Learning about theories does not only satisfy my curiosity, but also helps me distinguish between truth and rumour. Science subjects inspire me to think critically and I can apply science theories in current issues.

After scrolling down:



natural phenomena and how machines work. These are things we see every day in daily life but never truly understand how they work. My curiosity makes me an active learner and a challenger. When I find out a new phenomenon or a new science theory, I will spend time to look into it. I was curious about things around me since I was a child. Growing up, I learn about science that can explain how things work. It is the reason why I am keen on studying science.

I am active in learning things that I am interested in. I spend time on reading science news and articles about physics. They are not only about the experiments and formulae which I have learned in school, but new and interesting facts about science. Also, I often watch inspiring videos in the Internet in my free time. Some people may think surfing the Internet is a waste of time. However, I think it is actually a good way of learning. I can learn about the newest technology developments in the Internet which are not yet updated in textbooks.

I gained valuable experience and knowledge from activities held by science society in school. I remember using solutions and glass to make a delicate artwork. I learned that acid can dissolve glass and it can be applied in art. This experience made me enthusiastic about chemistry. When I get more curious and enthusiastic about something, I become more self-motivated.

Learning about theories does not only satisfy my curiosity, but also helps me distinguish between truth and rumour. Science subjects inspire me to think critically and I can apply science theories in current issues. For example, there is controversy about the use of tear gas. As a

Switching to different screen

```
begin
    checkScreenWidthScreen(state);
    repeat
        case state of
            0 : importTextFileScreen(state);
            1 : mainScreen(state);
            2 : statsScreen(state);
            3 : findScreen(state);
            4 : settingsScreen(state);
        end;
    until state = -1;
end.
```

This is the main part of my program.

`state` is used to determine next screen.

When `state` equals -1, the program ends.

Getting the current width of the console window

```
function getWindowWidth : Integer;
var i : Integer;
begin
    GotoXY(1, 1);
    i := 1;
    while WhereX = i do
    begin
        Write('*');
        i := i + 1;
    end;
    ClrScr;
    getWindowWidth := i-1;
end;
```

If `WhereX` doesn't match the counter `i`, it means that the '*' is printed on the next line.

So the windows' width is equal to the value of counter `i` - 1.

Selecting different buttons by using the arrow keys

```
repeat
    c := ReadKey;
    if c = #0 then
    begin
        c := ReadKey;
        if (c = #72) or (c = #80) then
        begin
            case position of
                1 : ...
                2 : ...
            end;
            case c of
                #72 : if position = ... then
                        position := ...
                      else
                        position := position - 1;
                #80 : if position = ... then
                        position := ...
                      else
                        position := position + 1;
            end;
            case position of
                1 : ...
                2 : ...
            end;
        end
    end
until c = #13;
```

The value of `position` refers to different buttons.

`#72` means arrow up and `#80` means arrow down

Deselect the originally selected button.

Change the value of `position`.

Set the newly selected button to selected state.

When the enter key is pressed, the program advances to next stage.

Testing

Unit test 1 - testing function `countNoOfSentences()`

Test plan

The objective is to test if the function can count the number of sentences correctly. Valid cases include:

- ‘This is a toy. There is a boy.’
- ‘He is good, and she is nice.’

Boundary cases include:

- ‘Ah.’
- ‘See ya!’
- ‘My name is Issac!’

Test result

Case	Expected result	Actual result
‘This is a toy. There is a boy.’	2	2
‘He is good, and she is nice.’	1	1
‘Ah.’	1	1
‘See ya!’	1	1
‘My name is Issac!’	1	1

The actual results match the expected result, so the function `countNoOfSentences()` passes the test.

Unit test 2 - testing function `countNoOfWords()`

Test plan

The objective is to test if the function can count the number of words accurately. Valid cases include:

- ‘My name is Issac.’
- ‘There is a dog.’
- ‘If I were Sunny, I would become a billionaire.’

Boundary cases include:

- ‘Nah.’
- ‘Hay!
  New line 1.
  New line 2.’

Test result

Case	Expected result	Actual result
‘My name is Issac.’	4	4
‘There is a dog.’	4	4
‘If I were Sunny, I would become a billionaire.’	9	9
‘Nah.’	1	1
‘Hay! New line 1. New line 2.’	7	7

The actual results match the expected result, so the function `countNoOfWords()` passes the test.

Unit test 3 - testing function `posOfString()`

Test plan

The objective is to test if the function can get the position of the substring in the long string accurately. Valid cases include:

- Substring: ‘ab’, long string: ‘aabc’
- Substring: ‘e’, long string: ‘never’
- Substring: ‘new’, long string: ‘weneewen’

Boundary cases include:

- Substring: ‘ab’, long string: ‘abccab’
- Substring: ‘a’, long string: ‘aaa’

Test result

Case	Expected result	Actual result
Substring: ‘ab’, long string: ‘aabc’	[1]	[1]
Substring: ‘e’, long string: ‘never’	[1, 3]	[1, 3]
Substring: ‘new’, long string: ‘weneewen’	[]	[]
Substring: ‘ab’, long string: ‘abccab’	[0, 4]	[0, 4]
Substring: ‘a’, long string: ‘aaa’	[0, 1, 2]	[0, 1, 2]

The actual results match the expected result, so the function `posOfString()` passes the test.

Unit test 4 - testing searching of words or phrases

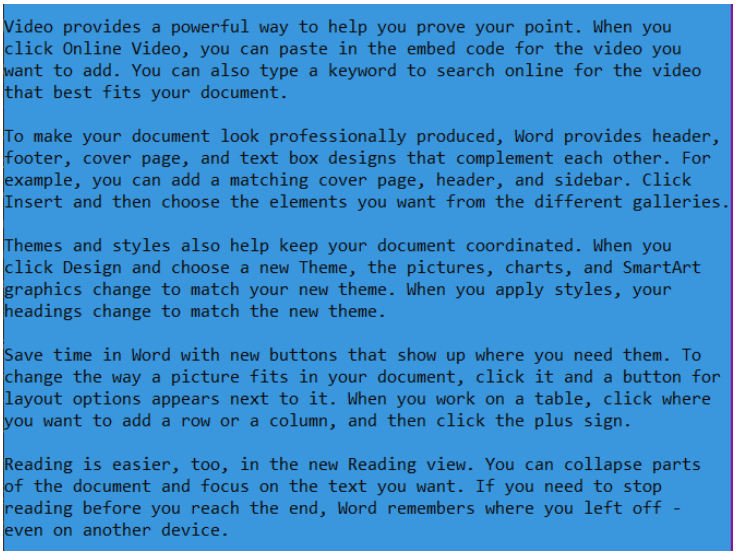
Test plan

The objective is to test if the targeted words or phrases can be searched and highlighted accurately. The testing composition is:
'Video provides a powerful way to help you prove your point. When you click Online Video, you can paste in the embed code for the video you want to add. You can also type a keyword to search online for the video that best fits your document.

To make your document look professionally produced, Word provides header, footer, cover page, and text box designs that complement each other. For example, you can add a matching cover page, header, and sidebar. Click Insert and then choose the elements you want from the different galleries.

Themes and styles also help keep your document coordinated. When you click Design and choose a new Theme, the pictures, charts, and SmartArt graphics change to match your new theme. When you apply styles, your headings change to match the new theme.
Save time in Word with new buttons that show up where you need them. To change the way a picture fits in your document, click it and a button for layout options appears next to it. When you work on a table, click where you want to add a row or a column, and then click the plus sign.

Reading is easier, too, in the new Reading view. You can collapse parts of the document and focus on the text you want. If you need to stop reading before you reach the end, Word remembers where you left off - even on another device.'



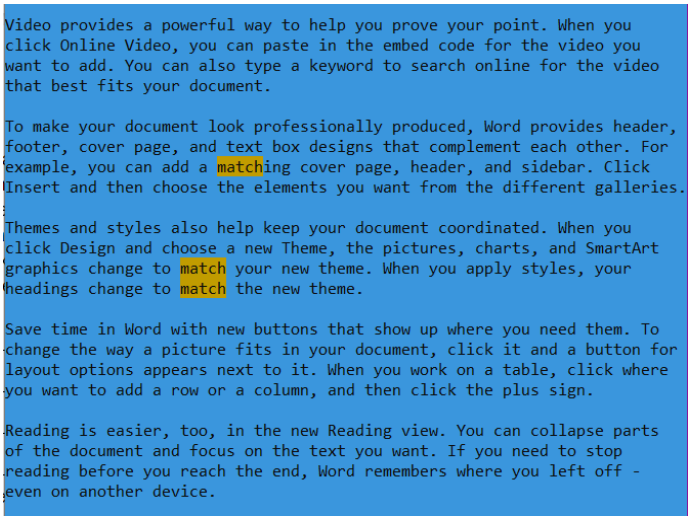
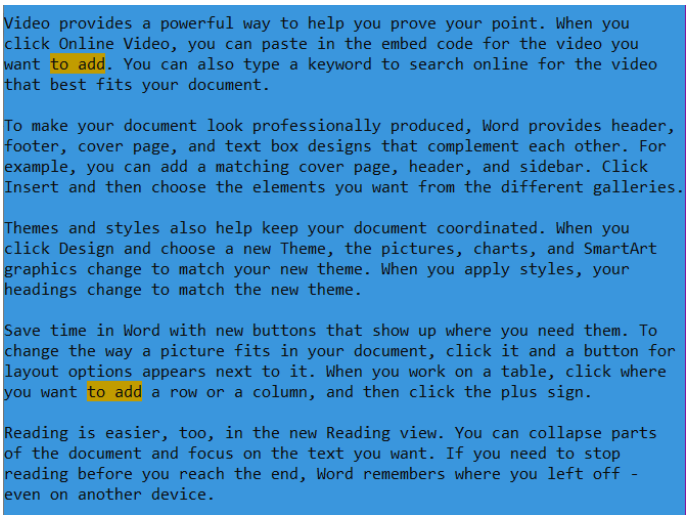
Valid search targets include:

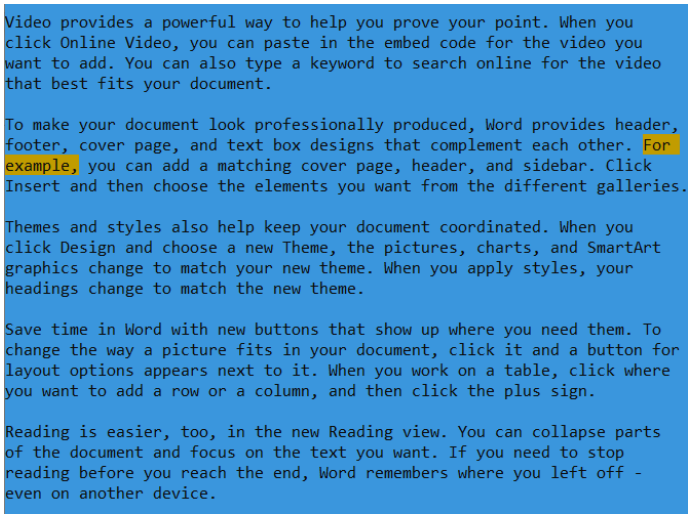
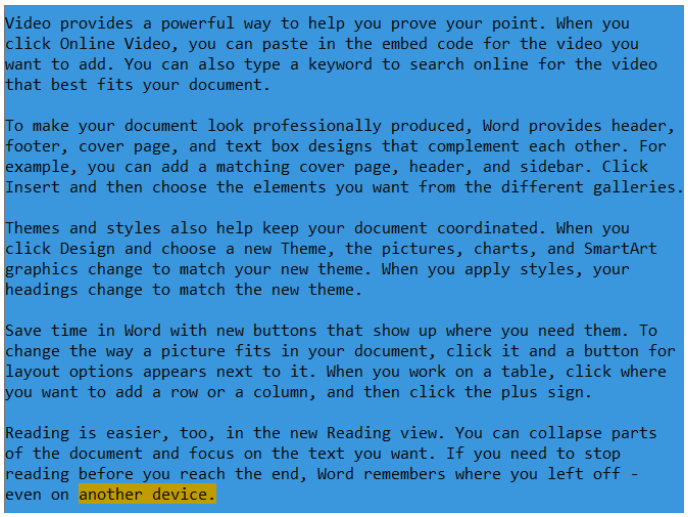
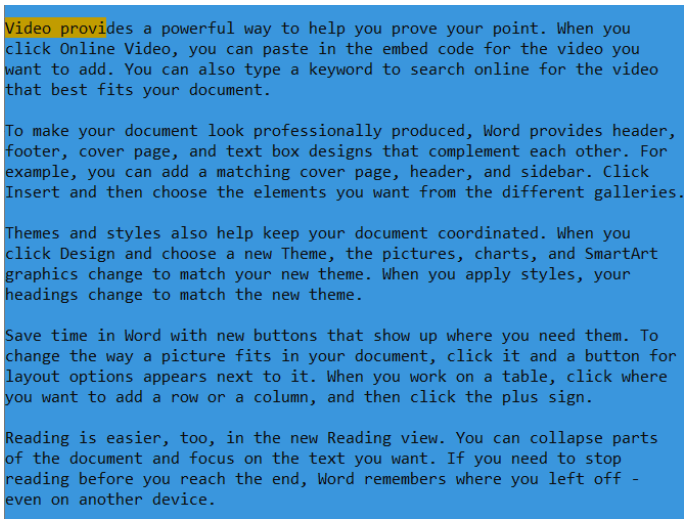
- ‘match’
- ‘to add’

Boundary search targets include:

- ‘For example,’
- ‘another device.’
- ‘Video provi’

Test result

Cases	Expected result	Actual result
‘match’	Total 3 words ‘match’ in paragraph 2 and 3 are highlighted.	
‘to add’	Total 2 phrases ‘to add’ in paragraph 1 and 4 are highlighted.	

‘For example,’	The phrase ‘For example,’ is highlighted in paragraph 2.	
‘another device.’	The phrase ‘another device.’ is highlighted in the last paragraph.	
‘Video provi’	The phrase ‘Video provi’ is highlighted in the first paragraph.	

The actual results match the expected result, so the searching function passes the test.

Unit test 5 - testing function `binarySearch()` in the custom type `stringList`

Test plan

The objective is to test if the function can return a boolean showing the existence of the search target in the string list. The string list must have been already sorted. The testing string list is `['Airplane', 'Chris', 'Egg', 'Isaac', 'Steven', 'Sunny', 'Zoo']`.

Some valid test cases include:

- 'Issac'
- 'Steven'
- 'Herny'

Some boundary cases include:

- 'airplane'
- 'zoo'

Test result

Cases	Expected result	Actual result
'Issac'	`FALSE`	`FALSE`
'Steven'	`TRUE`	`TRUE`
'Herny'	`FALSE`	`FALSE`
'Airplane'	`TRUE`	`TRUE`
'Zoo'	`TRUE`	`TRUE`

The actual results match the expected result, so `binarySearch()` passes the test.

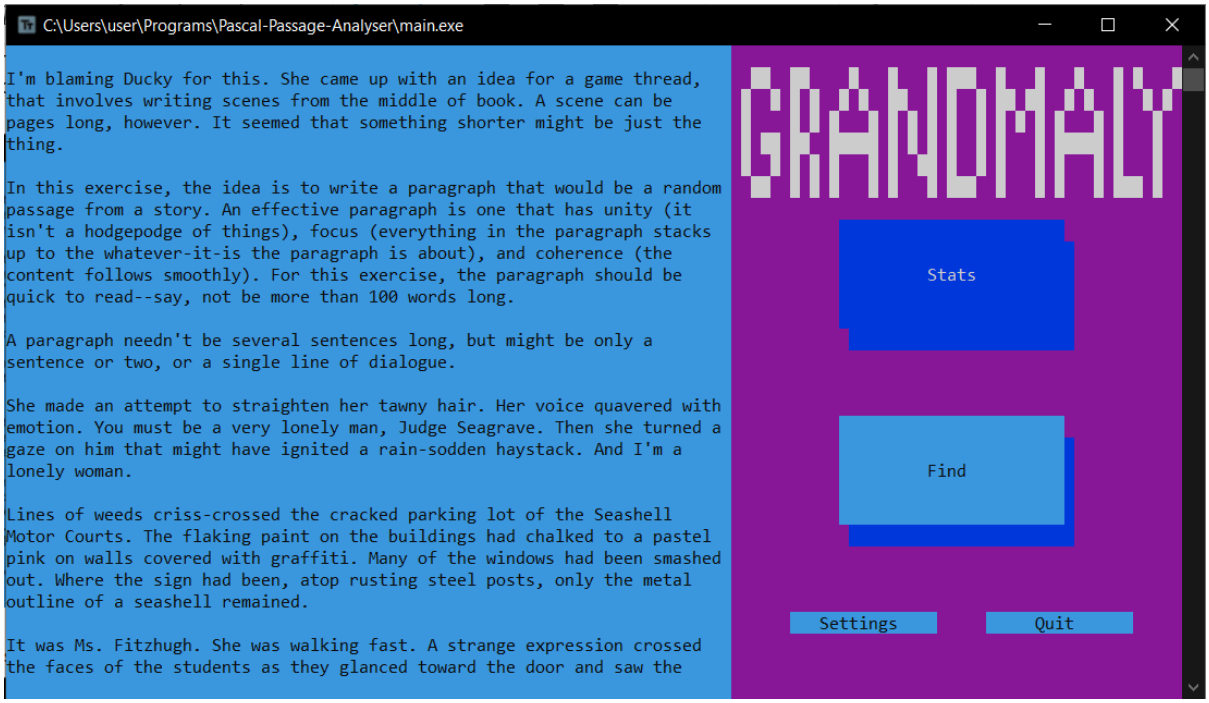
System test

Test plan

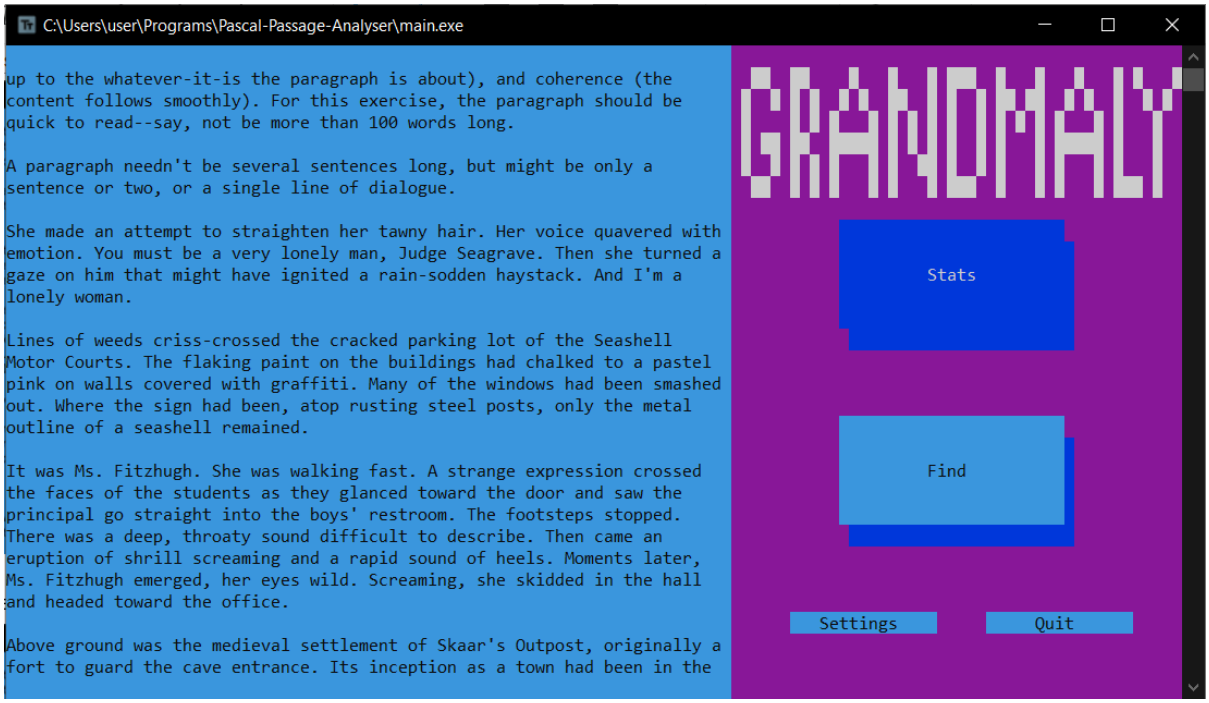
All of the functions and modules are integrated together into one main program, and then it is tested thoroughly.

Test result

The program runs without any logic and runtime error. All modules work as intended. During transition of different screens, there is no conflict between procedures and functions. No error exists between modules is observed. The program seems to be flawless.

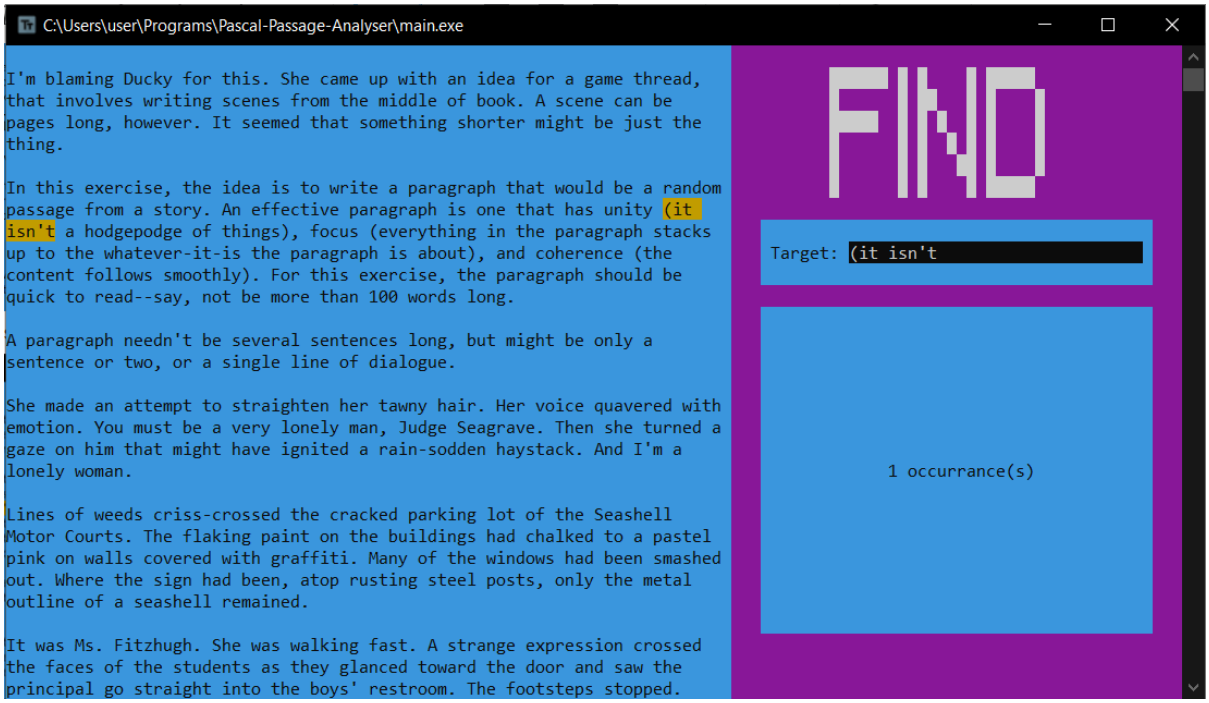


This is the main screen.

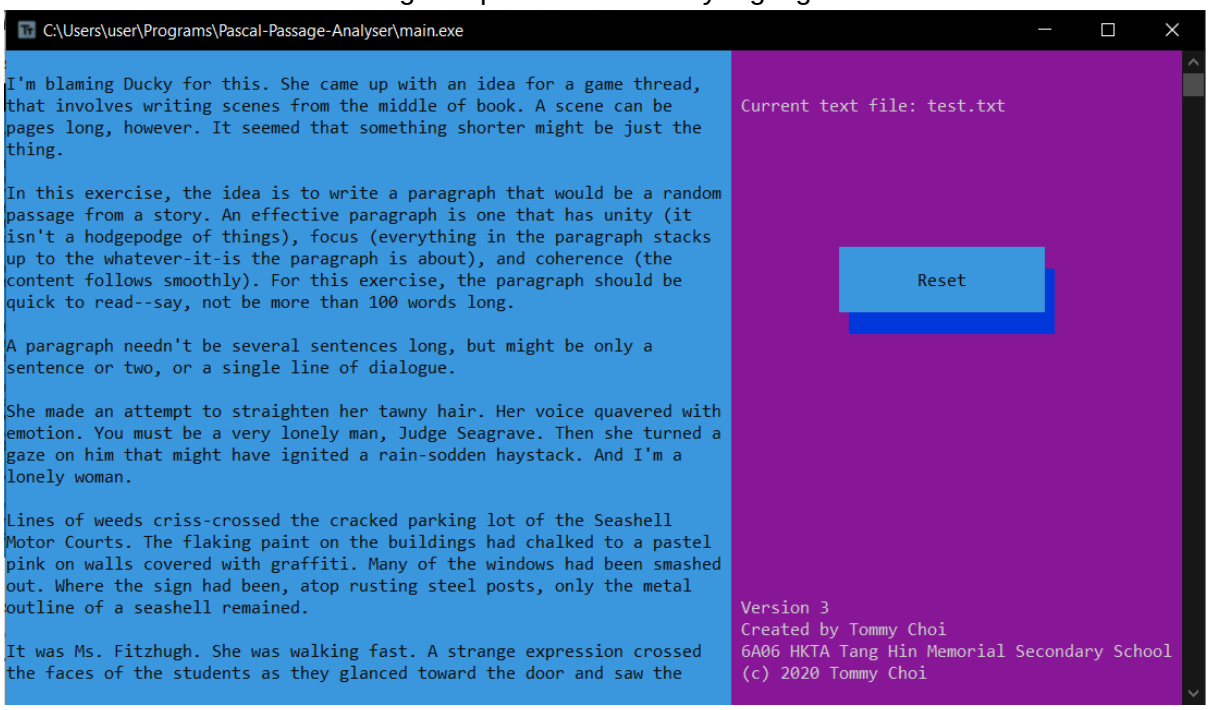


The passage is scrolled downward.

This is the statistics screen.



This is the find screen. The targeted phrase is correctly highlighted.



This is the settings page. The text file can be reset and import another composition to the program.

Evaluation

Things that learned

This is my first time doing a coding project. Through coding, I look through different documentation of Pascal. I found that I have just learned the tip of the iceberg. There are many things that I have not learned about Pascal in lessons. First, I learned that there is `set` in Pascal. A `set` can be defined by `[]`. It helps me a lot with the project. I have defined a constant set, storing a set of valid characters that users can type in some cases. For example, `valid_file_name_char = [#32..#126]` can perform a validation check for the file name that users typed. In many of the conditions in my program, the syntax `a in []` helps reduce code length. Originally I have to write so many cases like `(c >= 'A')and(c <= 'Z')and…` in the condition of `if`. It also improves code readability and maintainability.

Also, I learned the whole process of software development lifecycle. First, I prepared a plan for my project. Second, I did an analysis on the composition analyzer. Third, I designed the whole program structure and the UI. Forth, my algorithms, UI and program structure were implemented. Fifth, multiple units were tested separately and the whole integrated program was tested finally. I started doing documentation when I was implementing the program.

Room for improvement

The features in my program is not sufficient for analyzing a composition. Grammar checking, word collocation, synonyms and spell checking can be added to the composition analyzer. A word bank can be introduced into the program. It can be used to check spellings, and more sets of information can be generated, including the accuracy, frequency of wrong spelling, etc.

Also, in my `find` function, I failed to implement scrolling. In the other part of my program, the displayed composition can be scrolled using the page up and page down key. But in the `find` function, the composition is printed on the console simply, only allowing user to drag the console scroll bar to see the whole composition.