```cpp
#include "pch.h"
#include "CTextLineObject.h"

// Public constructors
CTextLineObject::CTextLineObject(CRect bounds, CString ID, std::vector<int> lineNums, CString text, BOOL active)
    : CAppObject(bounds, ID, active)
{
    this->text = text;

    this->highlighter.Y = bounds.top;
    this->highlighter.Height = bounds.Height();
    this->highlight = FALSE;
    hStart = 0;
    hEnd = 0;

    this->numSubLines = lineNums.size();
    this->lineNums = lineNums;

    this->smartColour = TRUE;
}
CTextLineObject::~CTextLineObject()
{
}

// Public commands
int CTextLineObject::draw(CDC* pDc, CSize textExtent, std::vector<vector<int>> brackets, int returnNewLines, BOOL printing, int printAreaLength)
{
    // Local variables
    BOOL autoBrkt = brackets.size() > 0; // If we have any auto brackets (used when not printing)
    int lengthPrint = 0; // Stores the current length of the print on the line that is currently being drawn to (only used when printing)

    // Gdiplus::Bitmap newLineArrow(L"res/newLine.png"); // Taking this out as it does not work and is not needed
    int printMaxCharLength = (printAreaLength - this->bounds.left * 2)/textExtent.cx;

    // If we art using SmartColour
    if (this->smartColour) {

        // Local variables
        CString word = L""; // Stores the word that is being found and is being colour checked
        CString character = L""; // Stores any characters that should seperate words
        int pos = 0; // Stores the position that is being drawn at
        BOOL comment = FALSE; // Stores whether the rest of the line is a comment
        int i = 0; // iterator for the loop, letter position that is being looked at

        // Iterate through characters in the line
        while (i < this->text.GetLength() + 1 && !comment) {

            // Checks to see if there is a comment
            if (this->text.Mid(i, theApp.commentType.GetLength()) != theApp.commentType) {

                // Character of the current position of the loop
                character = this->text.Mid(i, 1);
                BOOL hit = FALSE;

                // Checks to see if the character is a 'break' character - ie, if the character should seperate words - or if we are at the end of the loop
                if (character == L" " ||
                    character == L"!" ||
                    character == L"£" ||
                    character == L"$" ||
                    character == L"%" ||
                    character == L"^" ||
                    character == L"&" ||
                    character == L"*" ||
                    character == L"(" ||
                    character == L")" ||
                    character == L"+" ||
                    character == L"-" ||
                    character == L"/" ||
                    character == L":" ||
                    character == L";" ||
                    character == L"=" ||
                    character == L"<" ||
                    character == L"," ||
                    character == L"_" ||
                    character == L"-" ||
                    character == L">" ||
                    character == L"." ||
                    character == L"?" ||
                    character == L"/" ||
                    character == L"@" ||
                    character == L"{" ||
                    character == L"[" ||
                    character == L"|" ||
                    character == L"\\" ||
                    character == L"}" ||
                    character == L"]" ||
                    i == this->text.GetLength()) {

                    BOOL newColour = FALSE; // Stores if we have changed the colour, and if it needs to be changed back

                    // Checks if the word is a number
                    if (this->onlyNums(word)) {
                        // If so, sets the word to the number colour
                        pDc->SetTextColor(theApp.numberColour);
                        newColour = TRUE;
                    }
                    else { // If the word is not a number colour

                        // Binary search through all of the words to find if the current word matches one in the list
                        int it = 0, n = theApp.smartColour_String.size();

                        while (!newColour && it < n) {
                            // Binary Search
                            int it2 = 0, n2 = theApp.smartColour_String[it].size();
                            int min = 0, max = (n2)-1;


                            while (!newColour && it2 < n2) {

                                if (max < min) {
                                    newColour == TRUE;
                                }

                                int guess = floor((min + max) / 2);

                                CString tempWord = word;
                                CString tempAppWord = theApp.smartColour_String[it][guess];

                                tempWord.MakeLower();
                                tempAppWord.MakeLower();

                                // If this word on the list matches our searching word
                                if (tempWord == tempAppWord) {
                                    newColour = TRUE;
                                    pDc->SetTextColor(theApp.smartColour_Colour[it]); // Set the word's colour
                                }
                                else if (theApp.smartColour_String[it][guess] < word) {
                                    min = guess + 1;
                                }
                                else {
                                    max = guess - 1;
                                }
                                it2++;
                            }
                            it++;
                        }
                    }

                    // Drawing the word

                    // Drawing if we have a printer
                    if (printing) {

```

```cpp
                // Add the length if the word to the length of our print
                lengthPrint += word.GetLength();

                while (lengthPrint > printMaxCharLength) { // As a word could technically have a length greater than 50, we must use a while to check for lines

                    CString tempWord = L"";
                    if ((i - word.GetLength()) % printMaxCharLength != 0) {
                        tempWord = word.Mid(0, printMaxCharLength - ((i - word.GetLength()) % printMaxCharLength)); // Create a temporary word to store the start of the word that will go on the end of the current line
                        word = word.Mid(printMaxCharLength - ((i - word.GetLength()) % printMaxCharLength)); // Change the word to be whatever we didn't print and continue the loop
                    }
                    else {
                        pos++;
                    }

                    pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), tempWord, tempWord.GetLength()); // Print what we can to fill the line

                    lengthPrint -= printMaxCharLength; // Since we are creating a new line, subtract the old 60 characters from this

                    /*pos += tempWord.GetLength() + 2;

                    Gdiplus::Graphics g(pDc->GetSafeHdc());
                    Gdiplus::Rect expansionRect(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 + returnNewLines * this->bounds.Height(),
                        textExtent.cx, textExtent.cx);

                    g.DrawImage(&newLineArrow, expansionRect);*/

                    returnNewLines++;
                    pos = 0;
                }
                // Print the 'scraps' of the word at the end on the next line
                pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this-
>bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), word, word.GetLength());

                // Reset the position to draw from
                pos += word.GetLength();
            }
            // Not using printer
            else {
                pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2, word, word.GetLength());

                pos = i + 1;

                /*if (character != L"\"" && character != L"'") {
                    pos++;
                }*/
            }

            word = L""; // Reset the word since we are no longer using it

            // Now we print the character that we used to find the seperation

            BOOL grey = FALSE; // Stores if we used the autobracket

            // If there is a auto bracket for the character (never used when printing)
            if (this->active && autoBrkt && (character == L")" || character == L"}" || character == L"]") && this->brkPosContains(brackets, i) && !printing) {

                grey = TRUE;
                newColour = TRUE;

                pDc->SetTextColor(RGB(128, 128, 128)); // Set colour to grey for the auto bracket

                pDc->TextOut(this->bounds.left + 3 + (textExtent.cx * (i + word.GetLength())), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2, character, character.GetLength()); // Draw the auto bracket
            }

            // If, at any point, changed the colour
            if (newColour) {
                pDc->SetTextColor(RGB(0, 0, 0)); // Revert colour back to black (original colour)
            }

```

```
// If the character was not an auto bracket
if(!grey && character != L"\"" && character != L"'") {
    lengthPrint += character.GetLength();

    if (printing) {

        if (lengthPrint > printMaxCharLength) {


            pDc->TextOut(this->bounds.left + 2, this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2, character, character.GetLength());

            /*pos += character.GetLength() + 2;

            Gdiplus::Graphics g(pDc->GetSafeHdc());
            Gdiplus::Rect expansionRect(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 + returnNewLines * this->bounds.Height(),
                textExtent.cx, textExtent.cy);

            g.DrawImage(&newLineArrow, expansionRect);*/

            lengthPrint -= printMaxCharLength; // Since we are creating a new line, subtract the old 60 characters from this
            returnNewLines++;
            pos = 0;


        }
        else {

            pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), character, character.GetLength());
        }

        pos++;
    }
    else {

        pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * (i + word.GetLength())), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2, character, character.GetLength());
    }
}
```

```cpp
                }

                else if (character == L"\"" || character == L"'") {

                    pDc->SetTextColor(COLORREF(RGB(255, 128, 0)));

                    int j = i + 1;
                    BOOL found = FALSE;

                    while (!found && j < this->text.GetLength()) {

                        if (this->text.Mid(j, 1) == character) {

                            word = this->text.Mid(i, j - i + 1);

                            if (printing) {

                                // Add the length if the word to the length of our print
                                lengthPrint += word.GetLength();

                                while (lengthPrint > printMaxCharLength) { // As a word could technically have a length greater than 50, we must use a while to check for lines

                                    CString tempWord = L"";
                                    if ((i - word.GetLength()) % printMaxCharLength != 0) {
                                        tempWord = word.Mid(0, printMaxCharLength - ((i - word.GetLength()) % printMaxCharLength)); // Create a temporary word to store the start of the word that will go on the end of the current line
                                        word = word.Mid(printMaxCharLength - ((i - word.GetLength()) % printMaxCharLength)); // Change the word to be whatever we didn't print and continue the loop
                                    }
                                    else {
                                        pos++;
                                    }

                                    pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), tempWord, tempWord.GetLength()); // Print what we can to fill the line

                                    lengthPrint -= printMaxCharLength; // Since we are creating a new line, subtract the old 60 characters from this

                                    /*pos += tempWord.GetLength() + 2;
```

```

                                            Gdiplus::Graphics g(pDc->GetSafeHdc());
                                            Gdiplus::Rect expansionRect(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 + returnNewLines * this->bounds.Height(),
                                                textExtent.cx, textExtent.cx);

                                            g.DrawImage(&newLineArrow, expansionRect);*/

                                            returnNewLines++;
                                            pos = 0;
                                        }
                                    }

                                    // Print the 'scraps' of the word at the end on the next line
                                    pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), word, word.GetLength());

                                    pos += word.GetLength();
                                    found = TRUE;
                                }
                                j++;
                            }

                            if (!found) {
                                word = this->text.Mid(i);
                                pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), word, word.GetLength());

                            }


                            pDc->SetTextColor(RGB(0, 0, 0)); // Revert colour back to black (original colour)
                            i += word.GetLength() - 1;

                            word = L""; // Reset the word

                            pos = i + 1;

                        }

                        // If character is not a break away character, or we are not at the end of the loop
                        else {
```

```cpp
                    // Append the character to the word we are to look at

                    word += character;
                }
            }

            // This means that there is a comment
            else {

                // Set colour of the comment
                pDc->SetTextColor(theApp.commentColour);

                // Add the rest of the line to the word for drawing
                word.Append(this->text.Mid(i));

                // If we are printing...
                if (printing) {

                    // Add the length if the word to the length of our print
                    lengthPrint += word.GetLength();

                    while (lengthPrint > printMaxCharLength) { // As a word could technically have a length greater than 50, we must use a while to check for lines

                        CString tempWord = L"";
                        if ((i - word.GetLength()) % printMaxCharLength != 0) {

                            tempWord = word.Mid(0, printMaxCharLength - pos); // Create a temporary word to store the start of the word that will go on the end of the current line
                            word = word.Mid(printMaxCharLength - pos); // Change the word to be whatever we didn't print and continue the loop
                        }
                        else {
                            pos++;
                        }

                        pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), tempWord, tempWord.GetLength()); // Print what we can to fill the line

                        lengthPrint -= printMaxCharLength; // Since we are creating a new line, subtract the old 60 characters from this

                        /*pos += tempWord.GetLength() + 2;

                        Gdiplus::Graphics g(pDc->GetSafeHdc());
                        Gdiplus::Rect expansionRect(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 + returnNewLines * this->bounds.Height(),
                            textExtent.cx, textExtent.cy);

                        g.DrawImage(&newLineArrow, expansionRect);*/

                        returnNewLines++;
                        pos = 0;
                    }
                    // Print the 'scraps' of the word at the end on the next line
                    pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), word, word.GetLength());

                }
                // Not printing...
                else {
                    pDc->TextOut(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2, word, word.GetLength());
                }

                // Revert to base colour
                pDc->SetTextColor(RGB(0, 0, 0));

                // comment has been found, so break from loop
                comment = TRUE;
            }
            i++;
        }
    }
    // If we are not using SmartColour
    else {

        if (printing) {
            // Add the length if the word to the length of our print
            CString word = this->text;
            lengthPrint += word.GetLength();

            while (lengthPrint > printMaxCharLength) { // As a word could technically have a length greater than 60, we must use a while to check for lines

                CString tempWord = word.Mid(0, printMaxCharLength); // Create a temporary word to store the start of the word that will go on the end of the current line
                pDc->TextOut(this->bounds.left + 2, this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 +
                    returnNewLines * this->bounds.Height(), tempWord, tempWord.GetLength()); // Print what we can to fill the line

                /*int pos = tempWord.GetLength() + 2;

                Gdiplus::Graphics g(pDc->GetSafeHdc());
                Gdiplus::Rect expansionRect(this->bounds.left + 2 + (textExtent.cx * pos), this->bounds.top + this->bounds.Height() / 2 + returnNewLines * this->bounds.Height(),
                    textExtent.cx, textExtent.cy);

                g.DrawImage(&newLineArrow, expansionRect);*/

                word = word.Mid(printMaxCharLength); // Change the word to be whatever we didn't print and continue the loop
                lengthPrint -= printMaxCharLength; // Since we are creating a new line, subtract the old 60 characters from this
                returnNewLines++;
            }
            // Print the 'scraps' of the word at the end on the next line
            pDc->TextOut(this->bounds.left + 2, this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2 + returnNewLines * this->bounds.Height(), word, word.GetLength());
        }
        else {
            pDc->TextOut(this->bounds.left + 2, this->bounds.top + this->bounds.Height() / 2 - textExtent.cy / 2, this->text, this->text.GetLength());
        }
    }

    if (this->active && !this->highlight && !printing) {

        pDc->SelectStockObject(HOLLOW_BRUSH);
        pDc->Rectangle(bounds);
    }

    if (this->highlight && !printing) {

        this->highlighter.X = hStart * textExtent.cx + this->bounds.left + 2;
        this->highlighter.Width = (hEnd * textExtent.cx + this->bounds.left + 2) - this->highlighter.X;

        Gdiplus::Graphics g(pDc->GetSafeHdc());

        Gdiplus::SolidBrush solidBrush(Gdiplus::Color(100, GetRValue(theApp.highlightColour), GetGValue(theApp.highlightColour), GetBValue(theApp.highlightColour)));
```

```cpp
            g.FillRectangle(&solidBrush, this->highlighter);
        }
        return returnNewLines;
    }

    // Public getters & setters
    CString CTextLineObject::getText()
    {
        return this->text;
    }
    int CTextLineObject::getLength()
    {
        return this->text.GetLength();
    }
    void CTextLineObject::setText(CString text)
    {
        this->text = text;
    }
    void CTextLineObject::concatenateString(CString text, int position)
    {
        if (!GetKeyState(VK_INSERT)) {
            this->text = this->text.Mid(0, position) + text + this->text.Mid(position, this->text.GetLength());
        }
        else {
            this->text = this->text.Mid(0, position) + text + this->text.Mid(position + 1, this->text.GetLength());
        }
    }
    void CTextLineObject::backspace(int position)
    {
        if (!this->highlight) {
            this->text = this->text.Mid(0, position - 1) + this->text.Mid(position, this->text.GetLength());
        }
        else {
            this->text = this->text.Mid(0, min(this->hStart, this->hEnd)) + this->text.Mid(max(this->hStart, this->hEnd), this->text.GetLength());
            this->setHighlighting(FALSE);
        }
    }

    BOOL CTextLineObject::isHighlighting()
    {
        return this->highlight;
    }
    void CTextLineObject::setHighlighting(BOOL val)
    {
        if (!val) {
            this->highlighter.X = 0;
            this->highlighter.Width = 0;
            this->lineHighlight = FALSE;
        }
        this->highlight = val;
    }
```

```
void CTextLineObject::setHighlighter(int pos1, int pos2)
{
    if (pos1 > this->text.GetLength()) {
        pos1 = this->text.GetLength() + 1;
    }
    if (pos2 > this->text.GetLength()) {
        pos2 = this->text.GetLength() + 1;
    }

    if ((pos1 == 0 && pos2 == this->text.GetLength() + 1) || (pos2 
      == 0 && pos1 == this->text.GetLength() + 1)) {
        this->lineHighlight = TRUE;
    }

    this->hStart = min(pos1, pos2);
    this->hEnd = max(pos1, pos2);

    if (pos1 != pos2) {
        this->highlight = TRUE;
    }
}
void CTextLineObject::highlightLine()
{
    hStart = 0;
    hEnd = this->text.GetLength()+1;
    this->highlight = TRUE;
    this->lineHighlight = TRUE;
}

BOOL CTextLineObject::isLineHighlighted()
{
    return this->lineHighlight;
}

int CTextLineObject::getHStart()
{
    return this->hStart;
}

int CTextLineObject::getHEnd()
{
    return this->hEnd;
}

CString CTextLineObject::getHighlightedText()
{
    if (this->highlight) {
        return this->text.Mid(this->hStart, this->hEnd - this-
          >hStart);
    }
    return CString();
}

int CTextLineObject::getNumSubLines()
{
    return this->numSubLines;
```

```
}
std::vector<int> CTextLineObject::iGetLineNums()
{
    return this->lineNums;
}
CString CTextLineObject::sGetLineNums()
{
    CString num;
    for (auto& it : this->lineNums) {

        CString concat;
        concat.Format(L"%d", it);
        num.Append(concat);
        num.Append(L".");
    }
    return num.Mid(0, num.GetLength() - 1);
}
void CTextLineObject::addSublines(std::vector<int> subs)
{
    this->numSubLines += subs.size();
    for (auto& it : subs) {
        this->lineNums.push_back(it);
    }
}

void CTextLineObject::incrementLine(int subline, int val)
{
    if (subline > this->lineNums.size()) {
        throw("ERROR::subline::OUT OF RANGE");
    }
    this->lineNums[subline] += val;
}

void CTextLineObject::setBounds(CRect bounds)
{
    CAppObject::setBounds(bounds);

    this->highlighter.X = bounds.left;
    this->highlighter.Y = bounds.top;
    this->highlighter.Height = bounds.Height();
}

void CTextLineObject::move(int x, int y)
{
    CAppObject::move(x, y);
    this->highlighter.Offset(x, y);
}

BOOL CTextLineObject::onlyNums(CString str)
{
    for (int i = 0; i < str.GetLength(); i++) {

        if (str.Mid(i, 1) != L"0" &&
            str.Mid(i, 1) != L"1" &&
            str.Mid(i, 1) != L"2" &&
            str.Mid(i, 1) != L"3" &&
            str.Mid(i, 1) != L"4" &&
            str.Mid(i, 1) != L"5" &&
            str.Mid(i, 1) != L"6" &&
            str.Mid(i, 1) != L"7" &&
            str.Mid(i, 1) != L"8" &&
            str.Mid(i, 1) != L"9" ) {
            return FALSE;
        }
    }

    return TRUE;
}
BOOL CTextLineObject::brkPosContains(std::vector<std::vector<int>>   
  vector, int value, int side)
{
    for (int i = 0; i < vector.size(); i++) {

        if (vector[i][side] == value) {
            return TRUE;
        }
    }

    return FALSE;
}

```