* Raw text to - Component paragraphs, sentences & words *

```
struct word {
    char * data;
};
```

```
struct sentence {
    struct word * data;
    int word_count;
};
```

```
struct paragraph {
    struct sentence * data;
    int sentence_count;
};
```

```
struct document {
    struct paragraph *data;
    int paragraph_count;
};
```

Learning C is fun.

Learning pointers is more fun. It is good to have pointers.

① `struct sentence first_sentence_in_first_paragraph;`
`first_sentence_in_first_paragraph.data = {"Learning", "C", "is", "data"};`

② `struct paragraph first_paragraph;`
`first_paragraph.data = {{"Learning", .... , "fun"}};`

③ `struct sentence first_sentence_in_second paragraph;`
`first_sentence_in_second_paragraph.data = {"Learning", "pointers", ... "fun"};`

④ `struct sentence second_sentence_in_second_paragraph;`
`second_sentence_in_second_paragraph.data = {"It", ... , "pointers"};`

⑤ `struct paragraph second_paragraph;`
`second_paragraph.data = {{"Learning", "pointers", ... "fun"}, {"It", ... "pointers"}};`

⑥ `struct document doc;`
`doc.data = {{{"Learning", ... "fun"}}, {{"Learning", ... }, {"It", ... }}};`

Same thing - char*

Learning C is fun. In Learning pointers is more fun. It is good to have C.

Approach: Just give addresses

Sentence → going to have → strings!

* Void Initialise_document (Char * text) → Initialize Doc – Struct document.

* Struct paragraph Kth_Paragraph (Int K) → return Kth para

* Struct Sentence Kth_Sentence_In_mth_Paragraph (Int k, Int m) → kth Sent mth para!

* Struct Word Kth_word_In_m_th_Sentence_of_n_th_paragraph (Int k, Int m, Int n); → k th word, m th Sent, n th para.

I/p: Paragraph Count.

text (Char *)

Queries – number.

1 K : K th para

2 K m : K th Sentence m th para

3 K m n : K th word m th Sent n th para.

---

① Struct paragraph Kth_Paragraph (Struct document Doc, Int K)

```
{
return (Doc.data[K-1]);
}
```

② Struct Sentence Kth_Sent_m_th_para (Struct document Doc, Int k, Int m)

```
{
return (Doc.data[m-1].data[k-1]);
}
```

③ Struct Word Kth_word_In_mth_Sentence_of_nth_para (Struct document Doc, Int k, Int m, Int n)

```
{
return (Doc.data[n-1].data[m-1].data[k-1]);
}
```

---

case ① → '.' → next Sentence

'.' → nex Sentence

'\n' → next para.

Last line of the para (last) doesn't have \n.

① whenever ' ' → Create word

② whenever '.' → Create Sentence, Create word

③ whenever '\n', → Create para, Sentence, word.

'.\n' → no need for new Sentence → Create new para

make '.' → '\0' → end of String!

Same as previous one! – do with Struck

Problem faced: Created: `Struck document Doc;`

↓

Root cause!

Why? : Created in stack – dies when function gets done!

Solution: Use pointer & dynamic memory allocation!

```
Struck document get_document (char *text) {
    int para = 1, sent = 1, words = 1, i;
    Struck document *Doc;
    Doc = (Struck document *) malloc (sizeof (Struck document));
    Doc → data = (Struck paragraph *) malloc (sizeof (Struct paragraph));
    Doc → paragraph_count = 1;
    Doc → data [0]. data = (Struck Sentence *) malloc (sizeof (Struct
                                                        Sentence));

    Doc → data [0]. sentence_count = 1;
    Doc → data [0]. data [0]. data [0]. data = (char *) text;
    Doc → data [0]. data [0]. word_count = 1;

    for (i=0; text [i+1] != '\0' ; i++)
    {
        if (text [i+1] == '\n')
        {
            text [i] = '\0';
```

```
    p++;
}
switch(text[p])
{
    case '\n':
        para++;
        Doc->data = (struct paragraph*) realloc (Doc->data, para * sizeof
                                                 (struct paragraph));
        sent=0;
        Doc->data[para-1].sentence_count=0;
    case '.':
        sent++; (Doc->data[para-1].sentence_count)++;
        (Doc->data[para-1].data = (struct sentence*) realloc
            (Doc->data[para-1].data, sent * sizeof(struct sentence));
        words=0;
        Doc->data[para-1].data[sent-1].word_count=0;
    case ' ':
        words++;
        (Doc->data[para-1].data[sent-1].word_count)++;
        Doc->data[para-1].data[sent-1].data = (struct word*)
            realloc (Doc->data[para-1].data[sent-1].data,
                words * sizeof (struct word));
        Doc->data[para-1].data[sent-1].data[words-1].data =
                (char*) &text[p+1];
        text[p] = '\0';
    }
}
text[p] = '\0';
return (Doc.data[n-1].data[m-1].data[x-1]);
}
```

use: document * Doc

↳ Then allocate in heap