Algorithm 1: Extract thought phrases from the given sentences

```
Function getThoughts(response):
    subtrees = []
    thoughts = []
    types = clauses                                   // Only include sentence segments of these types.
    Trees = responseToTrees(response)          // Split response into multiple sentences (Implemented in [1]).
    for T ∈ Trees do
        if T is a sentence fragment then
            add phrases to types          // Sentence fragments are clauses, so include smaller sentence segments.
        for t ∈ T.subtrees do
            ℓ = t.leaves                                          // These are the words in the subtree.
            remove stop words and punctuation from ℓ
            // Exclude short sentence segments and sentence segments already added.
            if t ∈ types and t ∉ subtrees and |ℓ| ≥ 2 then
                subtrees.append(t)
                newThoughts = splitSubtree(t)
                thoughts.extend(newThoughts)
    if thoughts then
        return thoughts
    else
        remove punctuation from response
        return response

Function splitSubtree(subtree):
    words = subtree.words
    thoughts = []
    currentThought = []
    for i ∈ [0, |words|) do
        word = words[i]
        if word is an emoticon then
            continue
        if shouldSplit(words, i, currentThought, subtree.root) then
            thoughts.append(currentThought)
            currentThought = []
        else if word is not punctuation then
            currentThought.append(word)
    if currentThought then
        thoughts.append(currentThought))
    remove user-specified words from the beginning of each thought phrase
    return thoughts
```

REFERENCES

[1] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010

Function `shouldSplit`(*words*, *i*, *thought*, *root*)**:**
  if $i \geq |words| - 2$ **then return** False
  remove stop words from *thought*
  word = *words*[i]
  nextWord = *words*[i+1]
  smallRoots = phrases
  smallRootPhraseTypes = clauses and phrases
  $future$ = *words*[$i + 1$: end]
  nearFuture = $future$[0 : 3]
  thoughtBreaks = Coordinating conjunctions and Pentreebank punctuation types `:`, `,` and `.`
  $futureNoPunctuation$ = $future$ with punctuation removed
  $futureNoStopsNoPunctuation$ = $future$ with punctuation and stop words removed
  `// Do not split if the beginning part of the sentence segment is dependent on the next part.`
  if *word* = "," ***and*** *thought.head() is a preposition/subordinating conjunction* **then**
    return False
  if *[word* $\neq$ "," ***or*** $|futureNoStopsNoPunctuation| < 2$*]* ***and*** *[root* $\notin$ *smallRoots* ***and*** *(word is a coordinating conjunction* ***or*** *word* = ",") ***and*** $|thought| \geq 2$ ***and not*** *(word is in a dependent clause* ***xor*** *nextWord is in a dependent clause)]* **then**
    `// Check if the word separates a run on sentence.`
    right = *word*.rightSibling
    $left$ = *word*.leftSibling
    if *word.parent* $\in$ *smallRoots* **then**
      remove stop words and punctuation from *right*.words and $left$.words
      return *right* **and** $left$ **and** *right* $\in$ *smallRootPhraseTypes* **and** $left \in$ *smallRootPhraseTypes* **and** $|left$.words$| \geq 2$ **and** $|right$.words$| \geq 2$
    else
      return *right* **and** $left$ **and** *right* $\in$ *smallRootPhraseTypes* **and** $left \in$ *smallRootPhraseTypes*
  if *word* = "," ***and*** *nextWord is a coordinating conjunction* **then**
    stopsInNearFuture = *nearFuture*[1 : end] $\cap$ *thoughtBreaks*
  else
    stopsInNearFuture = *nearFuture* $\cap$ *thoughtBreaks*
  return *word* $\in$ *thoughtBreaks* **and not** *stopsInNearFuture* **and** $|thought| \geq 2$ **and** $|futureNoStopsNoPunctuation| \geq 2$ **and** $|futureNoPunctuation| > 3$ **and not** (*word* is in a dependent clause **xor** *nextWord* is in a dependent clause)