プログラミング演習I第6回レポート

学籍番号：2364902

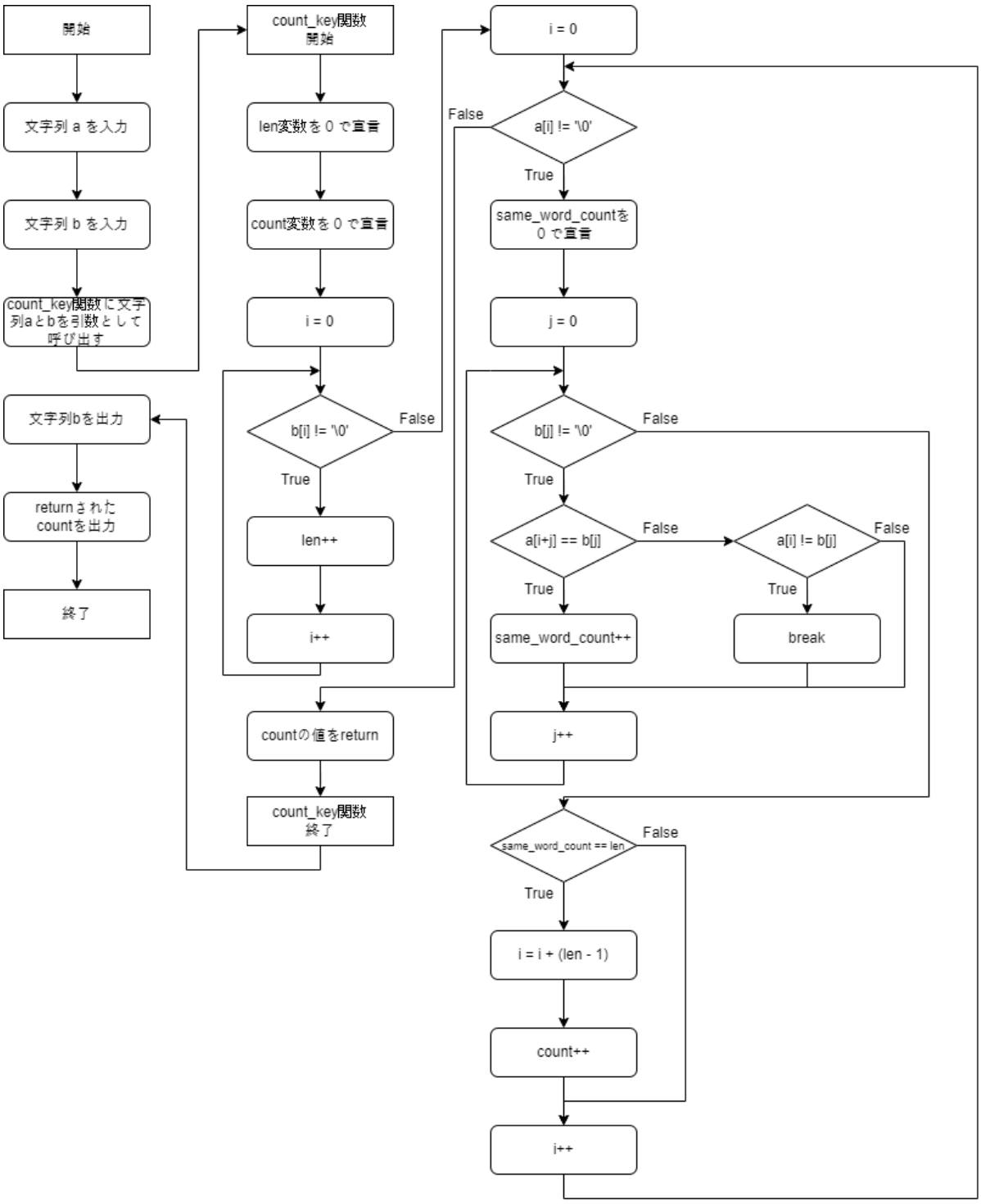
名前：キム　ギュソク

(1) 課題番号と課題内容

A. 基本課題 4

i. 文字列 a に、文字列 b が幾つ含まれているかを調べる関数 count_key() を作成する。文字列 a, b を引数とし、戻り値は含まれていた個数とする。「例 :int count_key(char a[], char b[])」main 関数では、キーボードから文字列 a および文字列 b を入力し、この関数を用いて、結果を表示する。

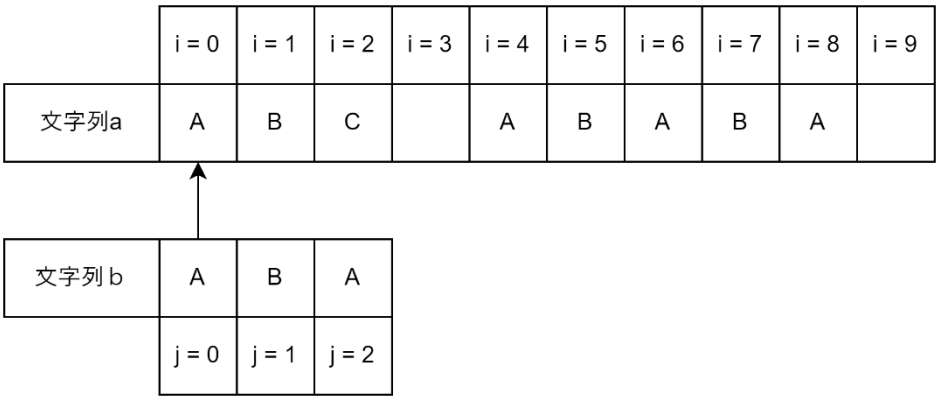
(2)フローチャートまたは疑似言語によるアルゴリズムの記述



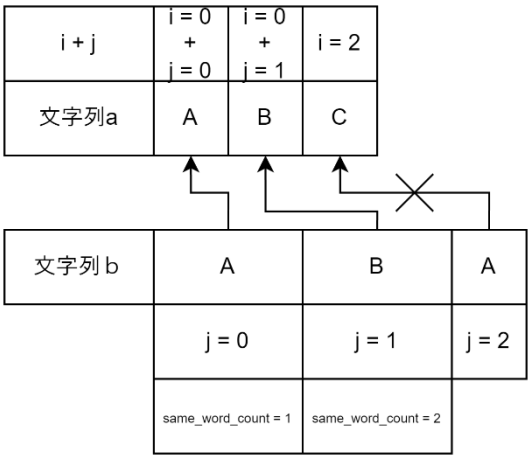
(3) アルゴリズムが正しいことの説明

i. 正当性

1. 二つの文字列aと b を入力して b の文字列がaに含まれているかを確認するプログラムである。フローチャートを見るとわかるように今回は同じ文字が同じ時にsame_word_count変数を増やすことでその連続で増えたsame_word_count変数が b 文字列の長さと同じ場合 b 文字列が存在するのでsame_word_countとlenを同じ時にcountを増やす操作を行うことでいくつか存在するか確認することができると思う。また、 b がABAの場合ABABAを2回数えることを防ぐためにカウントしたら同じ文字列が出た最後のインデックスに移動する必要がある。そのようなコントロールをするためにcountをする前にa配列のインデックスであるiをlen-1足すことで b 文字列が出たときに文字の最後のインデックスに移動することができる。
2. a = ABC ABABA CABAD ABA BAと b = ABAの場合
   A. この場合a文字列をインデックスを考えると

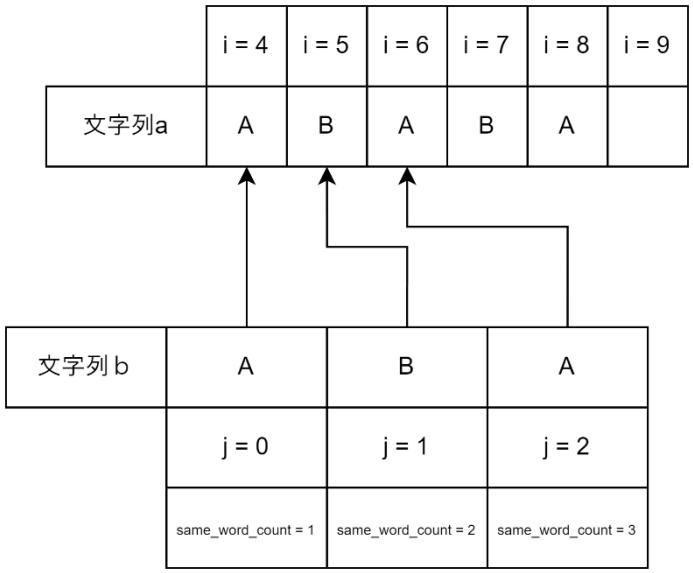


文字列 b の最初のインデックスの文字が文字列aに存在するか確認する。そして、見つかった場合は



このようにiの値は 0 のままでi+jを利用してまるでiと j が同時に増えな

がら同じであるか確認することができる。また、同じ時にsame_word_count＋＋することで何回連続で同じ文字が出たかを確認することができる。しかし、この方法で違う文字が出たときにｊの繰り返しを出てiは０のままである。重複を許すときにこの方法は正しいといえるが、この場合ABABAは一個として考えるので他の方法が必要である。
よって、インデクスiに同じ文字列が出たときにｂ文字列の長さを足すことで解決できる。例えば、次の場合



同じ文字列が出たのでcount++してiに戻るとi値は4であるため5からまた同じ文字を探索すると6~8のABAをもう一回カウントする。よって、ｊの繰り返しを脱出する際にiの4,5,6は確認済みなのでその後である7に移動するためにlen-1を足すことで解決できる。
よって、課題の要求する答えが出る正当なアルゴリズムであることが確認できる。

ii. 停止性

1. 今回の課題ではループを三つ使っている。
   A. ｂ文字列の長さを求めるためのループ
      i. ループの条件に文字列の最後が出ると脱出するように条件を付けて文字列のインデックスを増やした。このような操作を行うと必ず文字列最後ではループが停止するように設計されているので正しく停止する停止性を満足している。
   B. a文字列のループ
      i. このループも同じくa[i]が文字列の最後にたどり着くと停止するような条件の下でiを増やしているので無限の文字列にしない限り正しく停止するように出来ている。
   C. ｂ文字列のループ

i. このループもｂ文字列の長さを求めるためのループ同じなので正しく停止するように設計されていると言える。

(4) ソース・プログラムの説明

i. ソースコードの説明

1. 今回の課題では文字列の入力をgetsで入力するので文字列の最後に'¥0'が入れられるので#defineで'¥0'をEnd Of StrとしてEOSで宣言した。最大長さ100の二つの文字列配列を宣言する。その二つの文字列をcount_key関数に入れて呼び出してreturn値を%dとして出力する。key文字列であるｂ文字列の長さを入れるためのlen変数を０で初期化して宣言する。そして、繰り返し文を利用して文字列が終わるまでlen++することで文字列の長さを求める。その後、同じ文字が出た回数を入れるsame_word_countを０で初期化して宣言してｂ文字列の繰り返し文を作ってその中でa文字列とｂ文字列が同じ文字であるかを確認する。同じ場合はsame_word_count++する。そして、違う場合はｂ文字列を増やしなら確認する必要がないのでbreakして次のiに移動する。そして、ｂ文字列の確認が終わったらsame_word_countとlenが同じ場合ｂ文字列が存在するのでcount++する。同じ文字が出たところは再確認する必要がないので、確認が終わった後にインデクスiをlen-1だけ進める。そして、a文字列の全ての文字を確認したらcountをreturnする。特に、iを増やすたびにsame_word_countを初期化する必要があるのでｊの繰り返しを入る前に初期化する必要がある。

(5) 考察

A. 今回はｂ文字列を求めるためにfor文を利用して文字列が終わるまでlen変数を++することでｂ文字列の長さを求めたが、Ｃ言語のstring.hライブラリにはstrlenという関数が存在する。引数に文字列を入れると文字列の長さを返してくれる関数である。よって、文字列の長さを求めるときにはfor文を利用するよりstrlen関数を利用することがより可読性が優れたコードになると考えられる。また、string.hライブラリに含まれているstrstrという関数がある。この関数は一つ目の引数にポインタ変数を入れて二つ目の引数に探索したい文字列を入れると文字列が現れたポインタアドレスをreturnする関数である。この関数を利用すると複雑なコードをより簡略化することができると思う。より効率的なコードや早いコードであることを確認するためにはstring.hライブラリのstrstr関数の仕組みを確認する必要がある。しかし、既に作られている関数を利用することで可読性が上がる効率的なコードを作ることができると思う。例えば、stdio.hライブラリで提供しているprintf関数を全て自分で作った関数を利用するとそのコードは自分しか読めないコードになる可能性や他の人がコードを理解するときに多くの時間が必要であるため非効率的である。そして、iの繰り返しの中で同じであることを確認することなくjの繰り返しを始めるように作成されているが、この方法では違う場合でもbreakをする。一度繰り返し文に入って条件を確認するので余計な作業が生じます。このような余計な作業はコードの実行時間を遅くする要素なのでｂ文字列と同じ文字を見つけた時だけにｊの繰り返しを始まるように改善した方がより効率的だと考える。

(6) 感想

A. 同じ文字列が存在することを確認することで配列の概念とポインタの概念を理解するようになった。また、重複の処理を回避するために配列のインデックスの操作をどのように行うか考えることで配列の動きやポインタの動きについて理解することができた。