文字列処理

今回はＣ言語の文字列処理について復習し，文字列の探索手法について学びます．

文字列とは

プログラム上での文字の並びを表すのが文字列です．これは中身が空であっても同様に呼ばれます．Ｃ言語では"STRING"のように文字の並びを二重引用符"で囲んだものを文字列リテラルと呼びます．ASCII コードの場合，割り当てられる数値は図１のようになっています．

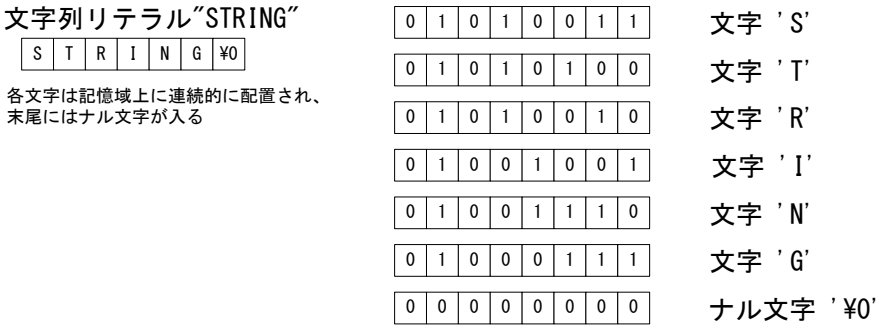


図１　文字列リテラル

各文字はメモリ上に連続して配置され，文字に対応する数値データによって管理されます．文字列リテラルには終端を示すためのナル文字が自動的に付加されます．ナル文字は文字コードによらず，全てのビットが 0 です．以下の宣言では文字列リテラルによって，ポインタ pt を初期化しています．この場合，文字列リテラル"1234"がメモリ上に確保されます．

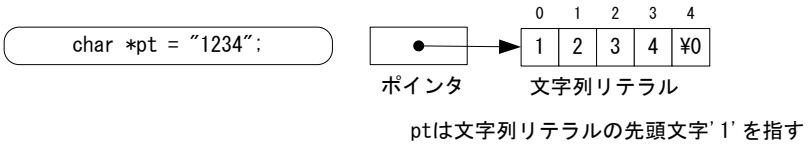


図２　ポインタと文字列リテラル

Ｃ言語では文字列リテラルは定数として扱われ，文字を再変更する際の動作は保証されていません．文字列を変更するならば，「char　st[5] = "1234";」のように配列に文字列を確保した上で行います．

文字列の長さ

配列を用いた文字列を利用することを考えます．文字列の最後には基本的にはナル文字が入っており，文字列の長さを求める場合，配列の先頭から始めて，ナル文字を線形探索していけばよいです．その際，ナル文字が見つかった添字が文字列の長さと一致します．

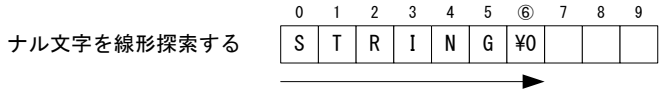


図３　文字列と長さ

文字列から文字を探索

文字列からナル文字ではなく，任意の文字を探索する手続きを考えます．例として文字列"SURROUND"から文字'R'を探索すると，図４のように，配列の先頭から線形探索を行うことになります．

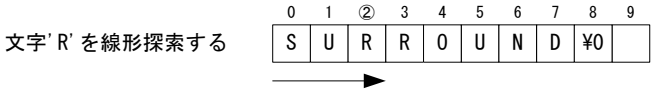


図４　文字の探索

文字列の比較

Ｃ言語では二つの文字列を比較するため,標準ライブラリの string.h において,strcmp 関数と strncmp 関数が提供されています.

◆strcmp 関数

この関数の仕様を以下に示す.

ヘッダ	#include <string.h>
形式	int strcmp(const char *s1, const char *s2);
解説	s1 が指す文字列と s2 が指す文字列の大小関係(先頭から順に 1 文字ずつ比較していき,異なる文字が出現したときに,それらの文字の対に成立する大小関係とする) の比較を行う.
返却値	等しければ 0, s1 が s2 より大きければ正の整数値, s1 が s2 より小さければ負の整数値を返す.

◆strncmp 関数

この関数の仕様を以下に示します.

ヘッダ	#include <string.h>
形式	int strncmp(const char *s1, const char *s2, size_t n);
解説	s1 が指す文字の配列と s2 が指す文字の配列の先頭 n 文字までの大小関係（先頭から順に 1 文字ずつ比較していき,異なる文字が出現したときに,それらの文字の対に成立する大小関係とする） の比較を行う.
返却値	等しければ 0, s1 が s2 より大きければ正の整数値, s1 が s2 より小さければ負の整数値を返す.

文字列探索（文字列から文字列を探索）

文字列探索とはある文字列の中に特定の文字列が含まれているかどうかを探索し，位置を特定する処理のことをいいます．たとえば，文字列"STRING"や"KING"から文字列"IN"を探す処理です．"IN"のように探し出そうとする文字列をパターンと呼び，探索する側の文字列をテキストと呼びます．ここでは以下の手法について学んでいきます.

・力まかせ法（単純法）
・KMP 法
・Boyer-Moore 法

力まかせ法（単純法）

力まかせ法はテキストの探索開始位置を１つずつ後ろにずらしていき，パターンとの間で 1 文字ずつ比較していきます．比較の途中で一致しない場合にはその後の比較は行いません．以下にテキスト"ABABCDEFGHA"からパターン"ABC"を探索する手続きを示します.

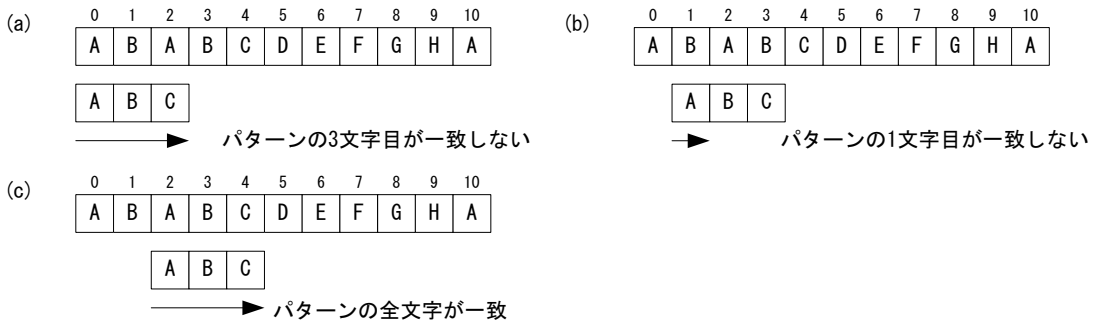


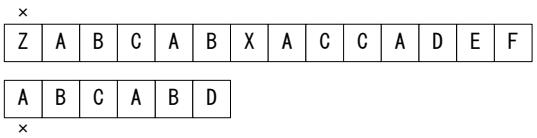
図５　力まかせ法の概略

図５（a）ではテキストの１文字目から比較を行い，パターンの３文字目が一致しません．(b) ではテキストの２文字目から比較を行い，パターンの１文字目が一致しません．(c) ではテキストの３文字目から比較を行い，全ての文字が一致して探索が成功します．

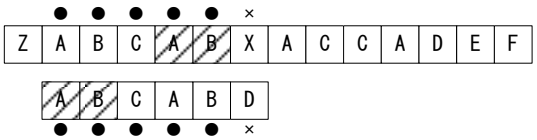
KMP 法

力まかせ法では文字が一致しない場合，パターンを移動してパターンの先頭から比較を行うことになるため，それまでの照合結果が捨てられてしまいます．それまでの照合結果を有効に利用しようとして考案されたのが KMP 法（Knuth-Morris-Pratt 法）です．

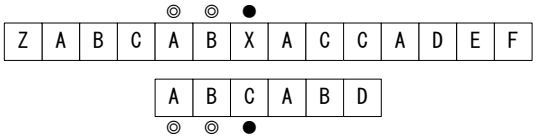
テキスト"ZABCABXACCADEF"からパターン"ABCABD"を探索するものとします．KMP 法ではまず，テキストとパターンの最初の文字から順に照合していきます．テキストの最初の文字'Z'はパターンの最初の文字'A'と一致しません．



次にパターンを１文字ずらし，再びパターンの先頭から順に照合を行っていくと，パターンの"ABCAB"まで一致した後，最後の文字'D'が不一致となります．



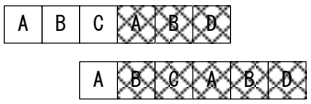
この場合，テキストとパターンの●が付いている部分で▨で示した"AB"が一致していることに着目し，この部分を照合済みとみなすと，テキスト側の文字'X'以降の部分がパターンの"CABD"と一致するかを調べればよいことになります．したがって，以下に示すようにパターンを３文字ずらし，３文字目の'C'から照合を再開することができます．



このように，KMP 法はテキストとパターン中の重なっている部分をうまく見つけ出して，照合し直す位置を求め，なるべくパターンの移動を大きくしようとするアルゴリズムです．実際に利用する場合には，パターン移動の際の何文字目から照合を再開するかについて，事前に表を作成しておき，効率を上げることになります．表を作成するためには，パターン内の重複した文字の並びを見つけなければなりません．パターンの１文字目が不一致の場合はパターンを１文字ずらして，先頭文字から照合を再開すればよいです．２文字目以降については以下のように行います．まず，パターン"ABCABD"どうしを１文字ずらして重ね合わせてみます．下の図で，白地で示した配列の部分に重なりは無いので，パターン移動時に先頭から照合をしなくてはならないことが分かります．そこで，２文字目の'B'の再開値を０とします．

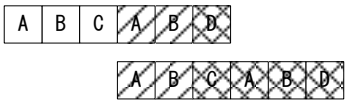
A	B	C	A	B	D
-	0				

さらに，パターンを１文字ずらします．ここでも文字は照合しないので，３文字目の'C'の再開値を０とします．



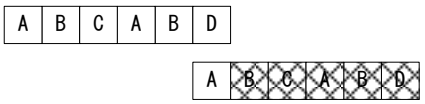
A	B	C	A	B	D
-	0	0			

さらに，パターンを 1 文字ずらすと，″AB″が照合します．そこで，これら二つの文字の再開値を 1 および 2 とします．



A	B	C	A	B	D
-	0	0	1	2	

次は，パターンを 2 文字ずらして比較を行います．結果，文字は一致しません．したがって，パターンの最後の文字'D'の再開値を 0 とします．



A	B	C	A	B	D
-	0	0	1	2	0

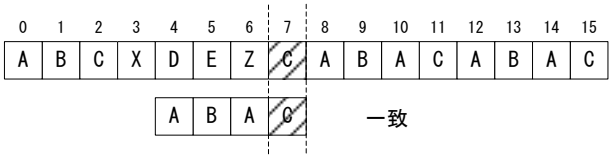
KMP 法の特徴はテキストを走査するカーソルが前進するだけになることです．これは力まかせ法には無い特徴です．しかし，このアルゴリズムは比較的複雑ですが，次に解説する Boyer-Moore 法と同等以下の性能しか発揮できないため，現実のプログラムで利用されることはあまりありません．

Boyer-Moore 法

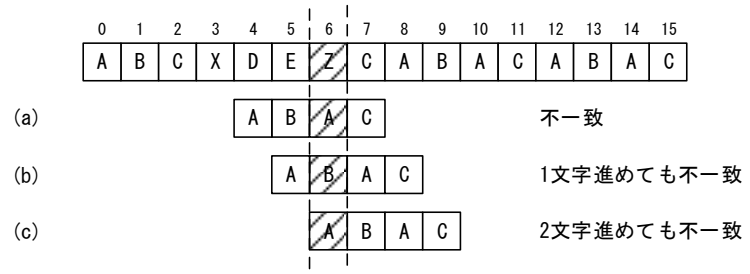
理論的にも実践的にも KMP 法より優れているのが BM 法（Boyer-Moore 法）です．力まかせ法や KMP 法とは異なり，照合をパターンの末尾文字から始めて先頭側へと進めていくという特徴があります．一致しない文字を見つけた場合には事前に用意した表に基づいて，その文字に応じたパターンの移動量を決定します．

テキスト″ABCXDEZCABACABAC″からパターン″ABAC″を探索する例で，このアルゴリズムについて考えます．探索開始時は，以下の図（a）のようにテキストとパターンの先頭文字を重ね，パターンの末尾文字'C'から照合します．照合するテキストの文字は'X'ですが，この'X'はパターンに存在しない文字です．そのため，図（b）～（d）のように 1～3 文字文移動しても一致しないことが分かります．

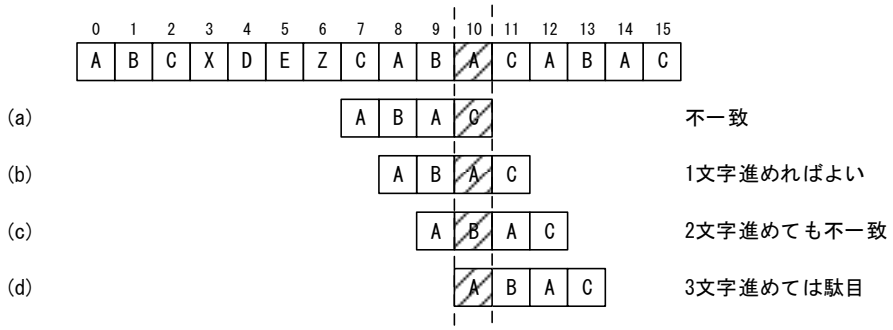
このように，パターンに含まれない文字をテキスト中に発見した場合には，そこまでの文字は全てスキップできることが分かります．そこで，次の照合の開始位置はパターンを一気に 4 文字後ろに移動します．



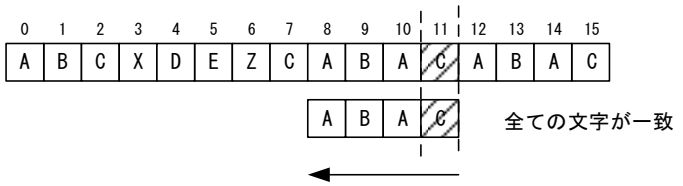
パターンの末尾文字'C'をテキストと比較すると，今度は一致しているので，次は一つ前の文字へ戻ります．



パターンの文字は'A'であり，テキストの'Z'とは一致しません．この場合，図（b），(c）のようにパターンを 1，2 文字移動しても，文字'Z'とは一致することはありません．よって，次の照合の開始位置はパターンを一気に 3 文字後ろに移動することができます．



図（a）に示すように，テキスト側の文字'A'はパターン末尾の文字'C'と一致しません．しかし，このテキストの文字'A'はパターン中の 1 文字目と 3 文字目の 2 箇所に含まれています．そこで，図（b）に示すように後ろ側の'A'が重なるようにパターンを 1 文字だけ移動します．この時，パターンの先頭側の'A'に重ねようとして一気に 3 文字移動しては上手くいきません．パターンを移動すると次の図のようになります．末尾側から順に文字を比較すると，すべて一致するので，これで探索は成功です．



探索成功

このアルゴリズムを利用するためには，各文字に出会ったときに，現在照合中の文字を何文字分ずらすかを事前に表にしておく必要があります．ここに示した例では次のスキップテーブルのようになります．

A	B	C	D	E	F	G	H	I	J	K	L	M
1	2	4	4	4	4	4	4	4	4	4	4	4

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
4	4	4	4	4	4	4	4	4	4	4	4	4

この表にない文字（小文字や数字など）についての移動量はすべて4

スキップテーブル

課題４

１．力任せ法，KMP 法，Boyer-Moore 法についてまとめてください．

２．力任せ法，KMP 法，Boyer-Moore 法の速度を測定します．その上で考察を行って下さい．速度の測定

については clock()は 1ms 程度の精度となっていますので，これ以上の測定時間となるよう，十分なサイズのテキストを用意して下さい．繰り返し処理を併用しても良いです．

参考：時間の測定プログラム

```
#include <stdio.h>
#include <time.h>    /* clock_t, CLOCKS_PER_SECを使うために必要 */

int main(void)
{
        clock_t begin, end;

        begin = clock();        /* 開始時間を記録 */
        func();                 /* 測定したいプログラムを書く */
        end = clock();          /* 終了時間を記録 */

        printf("測定時間 %f秒¥n",(float)(end-begin)/CLOCKS_PER_SEC);

        return 0;
}
```

clock_t clock(void)
clock はプログラム実行開始から経過したプロセッサ時間（エラー時は-1）を返します．
clock()/CLOCKS_PER_SEC は秒で表された時間です．使用するには time.h が必要です．