(Boyer-Moore algorithm)

a. Implement a C-function

```
int *lastOccurrence(char *pattern, char *alphabet) { ... }
```

that computes the last-occurrence function for the Boyer-Moore algorithm. The function should return a newly created dynamic array indexed by the numeric codes of the characters in the given alphabet (a non-empty string of ASCII-characters).

Ensure that your function runs in $O(m+s)$ time, where m is the size of the pattern and s the size of the alphabet.

Hint: You can obtain the numeric code of a `char c` through type conversion: `(int)c`.

a.

```
#define ASCII_SIZE 128

int *lastOccurrenceFunction(char *pattern, char *alphabet) {
   int *L = malloc(ASCII_SIZE * sizeof(int));
   assert(L != NULL);

   int i, s = strlen(alphabet);
   for (i = 0; i < s; i++)              // for all chars in the alphabet
      L[(int)alphabet[i]] = -1;         // ... initialise L[] to -1

   int m = strlen(pattern);
   for (i = 0; i < m; i++)
      L[(int)pattern[i]] = i;           // set L[]

   return L;
}
```

(Knuth-Morris-Pratt algorithm)

Develop, in pseudocode, a modified KMP algorithm that finds **every** occurrence of a pattern *P* in a text *T*. The algorithm should return a queue with the starting index of every substring of *T* equal to *P*.

Note that your algorithm should still run in *O(n+m)* time, and it should find every match, including those that "overlap".

Answer:

```
KMPMatchAll(T,P):
   Input  text T of length n, pattern P of length m
   Output queue with all starting indices of substrings of T equal to P

   F=failureFunction(P)
   i=0, j=0
   Q=empty queue
   while i<n do
      if T[i]=P[j] then
         if j=m-1 then
            enqueue i-j into Q     // match found
            i=i+1, j=F[m-1]        // continue to search for next match
         else
            i=i+1, j=j+1
         end if
      else
         if j>0 then
            j=F[j-1]
         else
            i=i+1
         end if
      end if
   end while
   return Q                        // if Q is empty, no match found
```