問題３　情報１

データの複雑さはデータエントロピーとよばれる。データエントロピーEは入力データに出現する単位データ（シンボルとよぶ）S_iの出現確率をP_iとして以下の式で求めることができる。

$$E = -\sum_i P_i \log_2 P_i$$

データエントロピーは情報の組合せ個数を表すための最少の平均ビット数を表している。この数値は整数とは限らないため、プログラム上では整数にしたビット数で実装する。

ここで、出現確率の偏りを利用して、データを圧縮・解凍するアルゴリズムを考える。データを圧縮する処理は符号化、解凍する処理は復号化ともよばれる。符号化において以下のようなアルゴリズムを考える。

（ステップ 1）入力データのシンボルごとの出現回数を求め、そのリストをつくる。

（ステップ 2）以下の手順で二分木をつくる。

(2-1) 二分木のルートノードを作成する。現在のノードをこのルートノードとする。

(2-2) 出現回数のリストにシンボルがある場合、現在のノードの左右の子ノードを作成する。出現回数のリストにシンボルがない場合、このステップを終了する。

(2-3) 出現回数のリストから出現回数の最も多いシンボルを取り出し、現在のノードの右の子ノードにする。

(2-4) 現在のノードを左の子ノードにし、(2-2) からを繰り返す。

（ステップ 3）（ステップ 2）の二分木のルートノードから検索し、以下の手順で入力データに出現するシンボルの符号を出力する。

(3-1) 入力データがなければ終了する。入力データがあれば入力データを取り出す。

(3-2) 現在のノードをルートノードとする。

(3-3) 現在のノードの右のノードにセットされたシンボルと入力のシンボルを比較し、合致しない場合、0 を出力して、左の子ノードに移動し、(3-3) を繰り返す。合致した場合は 1 を出力して

（3-1）から繰り返す。

（ステップ 4）二分木のデータ構造を出力する。

データの復号化は、符号化の際に出力された二分木のデータ構造を用いて、以下の手順で元のデータに戻すことができる。

（ステップ 1）現在のノードをルートノードとする。

（ステップ 2）入力データがなければ終了する。入力データがあれば、入力データから 1 ビット読み出し、1 の場合、現在のノードの右の子ノードにセットされたシンボルを出力し、（ステップ 1）から繰り返す。0 の場合、現在のノードを左の子ノードにし、（ステップ 2）を繰り返す。

上述の符号化・復号化のアルゴリズムの考え方をもとに、リスト 1 のように C 言語のプログラムを実装した。ただし、encode 関数と decode 関数は、main 関数に記述されているように呼び出され、decode 関数での sym_sort は encode 関数で作られたものが再利用されるとする。

以下の問題に答えなさい。ただし、$\log_2 7 = 2.80$とし、シンボルはアルファベット 1 文字であり、圧縮前は 8 ビットで表されるとする。

（1）以下に示す入力データについて、データエントロピーEを四捨五入して有効数字 2 桁で求めなさい。

ABACBBB

（2）（1）の入力データを上述のアルゴリズムで圧縮する場合の二分木を示しなさい。

（3）（1）の入力データを上述のアルゴリズムで圧縮した場合のビット列を求めなさい。

（4）（3）で圧縮されたビット列について、シンボルあたりの平均ビット数を四捨五入して有効数字 2 桁で求めなさい。

（5）上述のアルゴリズムにより入力シンボル列のビット数を削減できる理由を答えなさい。

(6) 以下のシンボル列を圧縮した場合、リスト 1 の(X)の時点での sym_sort の要素をインデクスの小さい順にすべて答えなさい。

ABACAABBCDEABBCCADEDCDCC

(7) リスト 1 の(Y)に示す比較操作は、n個のシンボルからなる入力データを符号化する場合、(X)に到達するまでに、最悪で何回行われるか答えなさい。

(8) 上述のアルゴリズムの二分木を作らずに、sym_sort の要素の並びを利用することで、二分木を用いる場合と同じように符号化・復号化できる。リスト 1 中の①と②のコードブロックを完成させなさい。

※次ページから「リスト 1」があることに注意すること。

リスト 1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct frequency_list_type {
 char symbol;
 int frequency;
 struct frequency_list_type *next;
};

struct frequency_list_type *frequency_list;

char *sym_sort;
int num_symbols;
char comp_code[1000];

void encode (char str[]){
 int str_len = strlen(str);
 frequency_list = NULL;
 struct frequency_list_type *curr, *prev;
 num_symbols = 0;

 int i,j;
 for(i=0;i<str_len;i++){
  if(frequency_list == NULL){
   frequency_list =
       (struct frequency_list_type *)malloc(sizeof(struct frequency_list_type));
   num_symbols ++;
   frequency_list->symbol = str[i];
   frequency_list->frequency = 1;
   frequency_list->next = NULL;
  }
  else {
   curr = frequency_list;
   while(1){
    if(curr->symbol == str[i]){
     curr->frequency ++;
     break;
    }
    if(curr->next != NULL) curr = curr->next;
    else{
     curr->next =
             (struct frequency_list_type *)malloc(sizeof(struct frequency_list_type));
     num_symbols ++;
     curr = curr->next;
     curr->symbol = str[i];
     curr->frequency = 1;
     curr->next = NULL;
     break;
    }
   }
  }
 }
 sym_sort = (char *)malloc(sizeof(char)*num_symbols);
 char max_sym;
 int max_freq;
 struct frequency_list_type *max_curr, *max_prev;
```

```
 i = 0;
 while(frequency_list != NULL){
  curr = frequency_list;
  prev = frequency_list;
  max_freq = curr->frequency;
  max_sym = curr->symbol;
  max_curr = curr;
  max_prev = prev;
  while(curr->next != NULL){
   prev = curr;
   curr = curr->next;
   if(curr->frequency > max_freq){    ・・・・・・(Y)
    max_freq = curr->frequency;
    max_sym = curr->symbol;
    max_curr = curr;
    max_prev = prev;
   }
  }
  sym_sort[i] = max_curr->symbol;
  i++;
  if(max_curr == frequency_list) frequency_list = max_curr->next;
  else max_prev->next = max_curr->next;
  free(max_curr);
 }
・・・・・・(X)
 int comp_code_count = 0;
 for(i=0;i<str_len;i++){
  for(j=0;j<num_symbols;j++){
   if(str[i] == sym_sort[j]){
    putchar('1');
    comp_code[comp_code_count] = '1';
    comp_code_count ++;
    break;
   }
   else{
    putchar('0');
    comp_code[comp_code_count] = '0';
    comp_code_count ++;
   }
  }
 }
 comp_code[comp_code_count] = '\0';
}

void decode(char str[]){
 int str_len = strlen(str);
 int i,j;
 j = 0;
 for(i=0;i<str_len;i++){
  if(str[i] == '0')[        ①        ]
  else{
   putchar(sym_sort[j]);
   [        ②        ]
  }
 }
}

int main(){
 char str[] = "ABACAABBCDEABBCCADEDCDCC";
 encode(str);
 decode(comp_code);
 printf("%s\n",str);
 return 1;
}
```