【解答】
［設問1］　a－ア，b－エ，c－イ，d－エ，e－エ
［設問2］　f－カ

【解説】

文字列の圧縮及び圧縮された文字列を復元するプログラムに関する問題である。問題文の量が多く，また，プログラムもやや長めであるため，解答に時間がかかった受験者も多いと思われる。圧縮及び復元の処理手順が示されているので，問題文の例を使って図を描きながらプログラムをトレースすれば解答は可能であり，根気よく丁寧に手を動かすことが大切である。解答が終わるまでの時間も考慮すると，難易度としては普通～やや難しいといえる。

はじめに，図1の例を使って圧縮する文字列をどのように探しているのかを確認する。問題文にあるように，圧縮後の文字数は3なので，一致する文字が4文字以上の場合に文字列を圧縮することになる。そのため，圧縮処理の対象となる文字の並び（以下，圧縮文字並び）の先頭位置（以下，圧縮文字位置）は，下図のように圧縮前の文字列の先頭から5文字目のEの位置となる。また，圧縮文字並びの比較対象とする文字の並び（以下，比較文字並び）の先頭位置（以下，比較文字位置）は，圧縮文字位置の4文字前のAの位置となる。圧縮文字並びの検索は，まず，圧縮文字位置と比較文字位置の二つの文字を比較し，圧縮文字位置の文字と一致する最初の1文字を探す。文字が一致しない場合は，圧縮文字位置は固定したまま，比較文字位置を一つずつ前に移動する。そして，一致する文字が見つかった場合は，圧縮文字位置と比較文字位置から一つずつ後ろに移動しながら，一致する文字数を数える。なお，文字列の先頭まで比較しても一致する文字が見つからない場合や移動した距離（圧縮文字位置からの距離）が26を超える場合，一致する文字が3文字以下の場合には，圧縮文字位置の文字は圧縮せずに配列 Compresseddata に格納する。

図1の例では，最初にEとAの比較を行うが，二つの文字は一致しない。そこで，比較文字位置を一つ前に移動することになるが，Aは配列の先頭であり，これ以上前に文字は存在しないため，検索を終了し，Eを配列 Compresseddata に格納する。

```
Plaindata  ABCDEFABCDABCDEF
           ↑   ↑
      比較文字位置 │
               圧縮文字位置
Compresseddata  ABCDE
  ※先頭から4文字は処理の開始時に格納済み
```

続いて圧縮文字位置，比較文字位置をそれぞれ一つ後ろに移動し，圧縮文字並びの検索を行う。最初に比較する文字はFとBであり，二つの文字は一致しないため比較文字位置を一つ前に移動し，FとAの比較を行う。この場合も文字は一致しないが，Aよりも前に文字列は存在しないので検索を終わり，Fを配列 Compresseddata に格納する。

```
Plaindata  ABCDEFABCDABCDEF
           ↑↑   ↑
           ②①  ※○付き数字は比較の順番を表す
Compresseddata  ABCDEF
```

続いて圧縮文字位置，比較文字位置をそれぞれ一つ後ろに移動し，AとCを比較する。先ほどと同様に，比較文字位置を一つずつ前に移動しながら二つの文字を比較していくと，3回目の比較で比較文字位置の文字がAになり文字が一致する。

```
Plaindata  ABCDEFABCDABCDEF
           ↑↑↑    ↑
           ③②①  ※○付き数字は比較の順番を表す
```

一致する先頭の文字が見つかったので，今度は圧縮文字位置，比較文字位置から一つずつ後ろに移動しながら順に文字を比較し，一致する文字数を数える。図1の例では，文字はA，B，C，Dと一致し，11文字目のAと5文字目のEを比較した際に不一致となり検索を終わる。

```
Plaindata  ABCDEFABCDABCDEF
           ↑↑↑↑↑ ↑↑↑↑↑
           ①②③④⑤ ①②③④⑤  ※○付き数字は比較の順番を表す
```

圧縮文字並びは，一致する文字数が最も多い比較文字並びとするため，今回の検索で対象となった比較文字並びの先頭よりも前に文字が存在する場合には，更に検索を続ける。図1の例の場合，最初に一致する文字は文字列の先頭のAであり，それより前に文字は存在しないので，1回の検索で圧縮文字並びが確定する。一致した文字数は4なので，圧縮列に変換して配列 Compresseddata に格納する。このとき，文字列は圧縮文字並びの先頭位置の6文字前から始まる文字列と4文字分一致するので，距離は6，文字数は4となる。ただし，距離と文字数はA～Zに置換えるため，6はF，4はDとなり，圧縮列は制御文字の$とF，Dの3文字となる。

```
Compresseddata  ABCDEF$FD
```

圧縮対象の文字はまだ残っているので，更に文字列の検索を行う。次に比較する圧縮文字位置と比較文字位置の文字は11文字目のAとそこから4文字前（7文字目）のAになる。最初の比較で文字が一致するので，圧縮文字位置，比較文字位置から一つずつ後ろに移動しながら，一致する文字数を数える。図1の例の場合，15文字目のEと11文字目のAの位置で文字が一致しなくなり，いったん検索を終わる。なお，一致した文字数は4，距離も4となる。

```
Plaindata  ABCDEFABCDABCDEF
                 ↑   ↑
               1文字目で一致

Plaindata  ABCDEFABCDABCDEF
                 ↑↑↑↑↑↑↑↑↑
                 ①②③④①②③④│
                     ↑
                     ⑤   ⑤  ※○付き数字は比較の順番を表す
```

先に説明したように，圧縮文字並びは，一致する文字数が最も多い比較文字並びとするため，直前の検索で対象となった比較文字並びよりも前に文字が存在する場合は更に検索を行う。このとき，一致した文字数が4以上であれば，これまでの検索で一致した最大の文字数を退避している変数 Maxfitnum と比較し，今回一致した文字数の方が大きければ，Maxfitnum と比較文字並びの先頭位置までの距離（移動文字数）の最大値を保持している変数 Maxdistance を更新する。

今回は，4文字が一致した比較文字並びよりも前に文字が存在するので，比較文字位置を1文字前のFの位置に移動（移動文字数を1増やす）して更に検索を行う。図1の例の場合，下図のようにFから順に一つずつ前に移動しながら一致する文字（A）を探す。

```
Plaindata  ABCDEFABCDABCDEF
           ↑↑↑↑↑↑    ↑
           ⑥⑤④③②①  ※○付き数字は比較の順番を表す
```

一致する先頭の文字が見つかれば，1文字ずつ後ろに移動しながら一致する文字数を数える。この検索では，文字列の先頭のAからFまでと一致するので，距離は10，文字数は6となる。なお，文字数のカウントは，圧縮文字並び側が文字列の最後に到達して終わる。

```
Plaindata  ABCDEFABCDABCDEF
           ↑↑↑↑↑↑    ↑↑↑↑↑↑
           ①②③④⑤⑥    ①②③④⑤⑥  ※○付き数字は比較の順番を表す
```

一致した文字数は6であり，その前の検索で一致した文字数の4より多いため，こちらを圧縮列とし，距離と文字数を変換した圧縮列$JFを配列 Compresseddata に格納する。

```
Compresseddata  ABCDEF$FD$JF
```

文字列の検索が配列の最後まで到達し，圧縮前の文字列は全て処理が終わったので，圧縮処理を終了する。

〔プログラム1〕

```
○副プログラム：Compress (文字型：Plaindata[]，整数型：Plength，
                         文字型：Compresseddata[]，整数型：Clength)
○文字型：Esym
○整数型：Pindex, Cindex
○整数型：Maxfitnum, Maxdistance, Distance, Fitnum
1  ▲Plength ≧ 1
2  │・Esym ← "$"     /* 制御記号の設定 */
3  │・Pindex ← 0     /* 圧縮前の文字列の文字位置を初期化 */
4  │・Cindex ← 0     /* 圧縮後の文字列の文字位置を初期化 */
5  │■(Pindex < Plength) and (Pindex < 4)
6  ││・Compresseddata[Cindex] ← Plaindata[Pindex]
7  ││・Cindex ← Cindex + 1
8  ││・Pindex ← Pindex + 1
9  │■
10 │■Pindex < Plength
11 ││・Maxfitnum ← -1
12 ││・Maxdistance ← -1
13 ││・Distance ← 4
14 ││■(Distance ≦ 26) and ( [   a   ] )
15 │││・Fitnum ← 0
16 │││/* 同じ文字並びの文字数を調べる */
17 │││■(Fitnum < Distance) and ((Pindex + Fitnum) < Plength)
18 ││││/* 文字が一致するかどうかを調べる */
19 ││││▲Plaindata[Pindex + Fitnum] ≠ [   b   ]
20 │││││・break  /* 最も内側の繰返し処理を終了する */
21 ││││▼
22 ││││・Fitnum ← Fitnum + 1
23 │││■
24 │││/* 文字数が4以上，かつ，最も多いかどうかを調べる */
25 │││▲(Fitnum ≧ 4) and (Maxfitnum < Fitnum)
26 ││││・Maxfitnum ← Fitnum
27 ││││・Maxdistance ← Distance
28 │││▼
29 │││・[   c   ]
30 ││■
31 ││▲Maxfitnum = -1
32 │││・Compresseddata[Cindex] ← Plaindata[Pindex]
33 │││・Cindex ← Cindex + 1
34 │││・Pindex ← Pindex + 1
35 ││├・Compresseddata[Cindex] ← Esym
36 │││・Compresseddata[Cindex + 1] ← IntToAlphabet (Maxdistance)
37 │││・Compresseddata[Cindex + 2] ← IntToAlphabet (Maxfitnum)
38 │││・Cindex ← Cindex + 3
39 │││・[   d   ]
40 ││▼
41 │■
42 │・Clength ← Cindex
43 ▼
```

行番号5～9：先頭からの4文字を配列 Compresseddata に格納
行番号14～30：圧縮文字並びの検索
　　圧縮文字並びの先頭の文字と一致する文字を探し，見つかれば一致する文字数を数える
行番号25～28：現時点の圧縮文字数と距離の最大値を退避
行番号32～34：圧縮しない1文字の場合の処理
行番号35～39：文字列を圧縮する場合の処理

問8　文字列の圧縮（データ構造及びアルゴリズム）　(H25 秋·FE 午後問 8)

【解答】

［設問1］　a－ア，b－エ，c－イ，d－エ，e－エ
［設問2］　f－カ

【解説】

文字列の圧縮及び圧縮された文字列を復元するプログラムに関する問題である。問題文の量が多く，また，プログラムもやや長めであるため，解答に時間がかかった受験者も多いと思われる。圧縮及び復元の処理手順が示されているので，問題文の例を使って図を描きながらプログラムをトレースすれば解答は可能であり，根気よく丁寧に手を動かすことが大切である。解答が終わるまでの時間も考慮すると，難易度としては普通～やや難しいといえる。

はじめに，図1の例を使って圧縮する文字列をどのように探しているのかを確認する。問題文にあるように，圧縮後の文字数は3なので，一致する文字が4文字以上の場合に文字列を圧縮することになる。そのため，圧縮処理の対象となる文字の並び（以下，圧縮文字並び）の先頭位置（以下，圧縮文字位置）は，下図のように圧縮前の文字列の先頭から5文字目のEの位置となる。また，圧縮文字並びの比較対象とする文字の並び（以下，比較文字並び）の先頭位置（以下，比較文字位置）は，圧縮文字位置の4文字前のAの位置となる。圧縮文字並びの検索は，まず，圧縮文字位置と比較文字位置の二つの文字を比較し，圧縮文字位置の文字と一致する最初の1文字を探す。文字が一致しない場合は，圧縮文字位置は固定したまま，比較文字位置を一つずつ前に移動する。そして，一致する文字が見つかった場合は，圧縮文字位置と比較文字位置から一つずつ後ろに移動しながら，一致する文字数を数える。なお，文字列の先頭まで比較しても一致する文字が見つからない場合や移動した距離（圧縮文字位置からの距離）が 26 を超える場合，一致する文字が3文字以下の場合には，圧縮文字位置の文字は圧縮せずに配列 Compresseddata に格納する。

図1の例では，最初にEとAの比較を行うが，二つの文字は一致しない。そこで，比較文字位置を一つ前に移動することになるが，Aは配列の先頭であり，これ以上前に文字は存在しないため，検索を終了し，Eを配列 Compresseddata に格納する。

```
Plaindata  ABCDEFABCDABCDEF
           ↑   ↑
      比較文字位置
               圧縮文字位置
Compresseddata  ABCDE
  ※先頭から4文字は処理の開始時に格納済み
```

続いて圧縮文字位置，比較文字位置をそれぞれ一つ後ろに移動し，圧縮文字並びの検索を行う。最初に比較する文字はFとBであり，二つの文字は一致しないため比較文字位置を一つ前に移動し，FとAの比較を行う。この場合も文字は一致しないが，Aよりも前に文字列は存在しないので検索を終わり，Fを配列 Compresseddata に格納する。

```
Plaindata  ABCDEFABCDABCDEF
           ↑↑   ↑
           ②①   ※○付き数字は比較の順番を表す
Compresseddata  ABCDEF
```

続いて圧縮文字位置，比較文字位置をそれぞれ一つ後ろに移動し，AとCを比較する。先ほどと同様に，比較文字位置を一つずつ前に移動しながら二つの文字を比較していくと，3回目の比較で比較文字位置の文字がAになり文字が一致する。

```
Plaindata  ABCDEFABCDABCDEF
           ↑↑↑   ↑
           ③②①   ※○付き数字は比較の順番を表す
```

一致する先頭の文字が見つかったので，今度は圧縮文字位置，比較文字位置から一つずつ後ろに移動しながら順に文字を比較し，一致する文字数を数える。図1の例では，文字はA，B，C，Dと一致し，11文字目のAと5文字目のEを比較した際に不一致となり検索を終わる。

```
Plaindata  ABCDEFABCDABCDEF
           ↑↑↑↑↑ ↑↑↑↑↑
           ①②③④⑤ ①②③④⑤   ※○付き数字は比較の順番を表す
```

圧縮文字並びは，一致する文字数が最も多い比較文字並びとするため，今回の検索で対象となった比較文字並びの先頭よりも前に文字が存在する場合には，更に検索を続ける。図1の例の場合，最初に一致する文字は文字列の先頭のAであり，それより前に文字は存在しないので，1回の検索で圧縮文字並びが確定する。一致した文字数は4なので，圧縮列に変換して配列 Compresseddata に格納する。このとき，文字列は圧縮文字並びの先頭位置の6文字前から始まる文字列と4文字分一致するので，距離は6，文字数は4となる。ただし，距離と文字数はA～Zに置換えるため，6はF，4はDとなり，圧縮列は制御文字の$とF，Dの3文字となる。

```
Compresseddata  ABCDEF$FD
```

圧縮対象の文字はまだ残っているので，更に文字列の検索を行う。次に比較する圧縮文字位置と比較文字位置の文字は11文字目のAとそこから4文字前（7文字目）のAになる。最初の比較で文字が一致するので，圧縮文字位置，比較文字位置から一つずつ後ろに移動しながら，一致する文字数を数える。図1の例の場合，15文字目のEと11文字目のAの位置で文字が一致しなくなり，いったん検索を終わる。なお，一致した文字数は4，距離も4となる。

```
Plaindata  ABCDEFABCDABCDEF
                 ↑   ↑
                 1文字目で一致

Plaindata  ABCDEFABCDABCDEF
                 ↑↑↑↑↑↑↑↑↑
                 ①②③④①②③④
                     ↑
                     ⑤   ⑤   ※○付き数字は比較の順番を表す
```

先に説明したように，圧縮文字並びは，一致する文字数が最も多い比較文字並びとするため，直前の検索で対象となった比較文字並びよりも前に文字が存在する場合は更に検索を行う。このとき，一致した文字数が4以上であれば，これまでの検索で一致した最大の文字数を退避している変数 Maxfitnum と比較し，今回一致した文字数の方が大きければ，Maxfitnum と比較文字並びの先頭位置までの距離（移動文字数）の最大値を保持している変数 Maxdistance を更新する。

今回は，4文字が一致した比較文字並びよりも前に文字が存在するので，比較文字位置を1文字前のFの位置に移動（移動文字数を1増やす）して更に検索を行う。図1の例の場合，下図のようにFから順に一つずつ前に移動しながら一致する文字（A）を探す。

```
Plaindata  ABCDEFABCDABCDEF
           ↑↑↑↑↑↑    ↑
           ⑥⑤④③②①   ※○付き数字は比較の順番を表す
```

一致する先頭の文字が見つかれば，1文字ずつ後ろに移動しながら一致する文字数を数える。この検索では，文字列の先頭のAからFまでと一致するので，距離は10，文字数は6となる。なお，文字数のカウントは，圧縮文字並び側が文字列の最後に到達して終わる。

```
Plaindata  ABCDEFABCDABCDEF
           ↑↑↑↑↑↑    ↑↑↑↑↑↑
           ①②③④⑤⑥    ①②③④⑤⑥   ※○付き数字は比較の順番を表す
```

一致した文字数は6であり，その前の検索で一致した文字数の4より多いため，こちらを圧縮列とし，距離と文字数を変換した圧縮列$JFを配列 Compresseddata に格納する。

```
Compresseddata  ABCDEF$FD$JF
```

文字列の検索が配列の最後まで到達し，圧縮前の文字列は全て処理が終わったので，圧縮処理を終了する。

〔プログラム1〕

```
○副プログラム：Compress（文字型：Plaindata[]，整数型：Plength,
                          文字型：Compresseddata[]，整数型：Clength）
○文字型：Esym
○整数型：Pindex, Cindex
○整数型：Maxfitnum, Maxdistance, Distance, Fitnum
1   ▲Plength ≧ 1
2   │・Esym ← "$"     /* 制御記号の設定 */
3   │・Pindex ← 0     /* 圧縮前の文字列の文字位置を初期化 */
4   │・Cindex ← 0     /* 圧縮後の文字列の文字位置を初期化 */
5   │■(Pindex < Plength) and (Pindex < 4)
6   ││・Compresseddata[Cindex] ← Plaindata[Pindex]
7   ││・Cindex ← Cindex + 1
8   ││・Pindex ← Pindex + 1
9   │■
10  │■Pindex < Plength
11  ││・Maxfitnum ← -1
12  ││・Maxdistance ← -1
13  ││・Distance ← 4
14  ││■(Distance ≦ 26) and ( [ a ] )
15  │││・Fitnum ← 0
16  │││/* 同じ文字並びの文字数を調べる */
17  │││■(Fitnum < Distance) and ((Pindex + Fitnum) < Plength)
18  ││││/* 文字が一致するかどうかを調べる */
19  ││││▲Plaindata[Pindex + Fitnum] ≠ [ b ]
20  │││││・break /* 最も内側の繰返し処理を終了する */
21  ││││▼
22  ││││・Fitnum ← Fitnum + 1
23  │││■
24  │││/* 文字数が4以上，かつ，最も多いかどうかを調べる */
25  │││▲(Fitnum ≧ 4) and (Maxfitnum < Fitnum)
26  ││││・Maxfitnum ← Fitnum
27  ││││・Maxdistance ← Distance
28  │││▼
29  │││・[ c ]
30  ││■
31  ││▲Maxfitnum = -1
32  │││・Compresseddata[Cindex] ← Plaindata[Pindex]
33  │││・Cindex ← Cindex + 1
34  │││・Pindex ← Pindex + 1
35  ││├・Compresseddata[Cindex] ← Esym
36  │││・Compresseddata[Cindex + 1] ← IntToAlphabet (Maxdistance)
37  │││・Compresseddata[Cindex + 2] ← IntToAlphabet (Maxfitnum)
38  │││・Cindex ← Cindex + 3
39  │││・[ d ]
40  ││▼
41  │■
42  │・Clength ← Cindex
43  ▼
```

行番号 5～9：先頭からの4文字を配列 Compresseddata に格納
行番号 14～30：圧縮文字並びの検索
　　圧縮文字並びの先頭の文字と一致する文字を探し，見つかれば一致する文字数を数える
行番号 25～28：現時点の圧縮文字数と距離の最大値を退避
行番号 32～34：圧縮しない1文字の場合の処理
行番号 35～39：文字列を圧縮する場合の処理