問8　簡易メモ帳のメモリ管理（データ構造及びアルゴリズム）（H28春-FE 午後問8）

【解答】

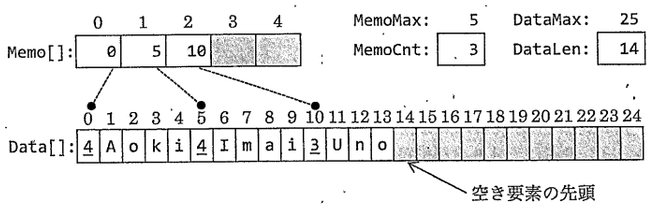
［設問1］　a－ア，b－ウ，c－エ，d－イ

［設問2］　e－ウ，f－カ，g－ケ

【解説】

簡易メモ帳のメモリ管理に関する問題である。難しそうなテーマであるが，内容は文字列の操作であり，配列の扱いを理解していれば対応できる。設問は，メモの編集処理（メモの追加・削除・変更・移動）を行うプログラムの空欄の穴埋めと参照されなくなったメモを取り除き，空き要素を増やすプログラムのトレース（追跡）の二つである。プログラムはいずれも短く，示された図で，配列を使ってどのようにデータを格納し，管理するのかを確認しながら処理をトレースすれば，確実に正解できる。

はじめに，メモの格納と管理をどのように行っているのかを確認する。メモの格納と管理には，2個の配列 Memo[ ], Data[ ]と4個の変数 MemoCnt, MemoMax, DataLen, DataMax を使用する。各メモは，メモの文字列の前に文字列の長さを付け加えて，Data[ ]の1要素に1バイトずつ格納する。そして，各メモの格納位置の情報（メモの文字長が格納されている Data[ ]の要素番号）を，メモの表示順に Memo[ ]の先頭の要素から順に設定して管理する。格納できるメモの最大件数（Memo[ ]の要素数）は MemoMax，現在格納されているメモの件数は MemoCnt に格納されている。また，メモを格納できる最大文字数（Data[ ]の要素数）は DataMax，Data[ ]に現在格納されている文字数は DataLen に格納されている。なお，DataLen は，Data[ ]に現在格納されている文字数を表すと同時に，Data[ ]の最初の空き要素の位置も表している。



メモの追加は，Data[ ]の最初の空き要素以降に格納する。メモの削除は，Data[ ]にはメモを残したまま，Memo[ ]から位置情報を削除することで行う。また，メモの変更は，変更前のメモは Data[ ]に残したまま，変更後のメモを Data[ ]の最初の空き要素以降に格納し，Memo[ ]の格納位置の情報を変更後の要素番号に変更することで行う。メモの移動は，Memo[ ]の要素の並び順を変えて，Memo[fromPos]の内容を Memo[toPos]の位置に移動する。このとき，まず Memo[fromPos]の要素を退避し，fromPos＜toPos の場合は，Memo[fromPos ＋ 1]～Memo[toPos]の要素を1要素ずつ前にずらし，fromPos＞toPos の場合は，Memo[fromPos － 1]から前方の Memo[toPos]までの要素を1要素ずつ後ろにずらしてから，退避した値を Memo[toPos]の位置に格納する。これらのことは，各関数の説明や関数を実行後の図から読み取ることができる。

続いて，各プログラムを確認する。

〔プログラム〕

○大域　整数型：MemoCnt, MemoMax, Memo[MemoMax]
○大域　整数型：DataLen, DataMax
○大域　8ビット論理型：Data[DataMax]

○関数：resetMemo()

・MemoCnt ← 0
・DataLen ← 0

resetMemo は，全てのメモを消去する関数である。MemoCnt と DataLen に 0 を設定することによって，Memo[ ]と Data[ ]の全要素を“空き”状態にする。

(行番号)

```
 1  ○関数：addMemo(整数型：textLen, 文字列型：text)
 2    ○整数型：i
 3    ・Memo[MemoCnt] ← [ a ]
 4    ・MemoCnt ← MemoCnt ＋ 1
 5    ・Data[DataLen] ← textLen
 6    ・DataLen ← DataLen ＋ 1
 7    ■ i：0, i ＜ textLen, 1
 8    │ ・Data[DataLen ＋ i] ← text[i]
 9    ■
10    ・DataLen ← [ b ]
```

addMemo は，1件のメモを追加する関数である。長さ textLen の文字列 text を Data[ ]の最初の空き要素以降に格納し，その格納位置の情報を Memo[ ]に設定する。

行番号3：追加したメモの格納位置の情報を Memo[ ]に設定
行番号4：メモの件数を一つ増やす
行番号5：Data[ ]の最初の空き領域に追加する文字列の長さを格納
行番号6：Data[ ]に格納されている文字数を一つ増やす
　　これによって，DataLen は文字列 text の先頭文字の格納場所を表すことになる
行番号7～9：文字列 text を1文字ずつ Data[ ]に格納
行番号10：Data[ ]に格納されている文字数をメモの文字数分増やす

```
21  ○関数：deleteMemo(整数型：pos)
22    ○整数型：i
23    ・i ← [ c ]
24    ■ i ＜ MemoCnt
25    │ ・Memo[i － 1] ← Memo[i]
26    │ ・i ← i ＋ 1
27    ■
28    ・MemoCnt ← MemoCnt － 1
```

deleteMemo は，1件のメモを削除する（表示の対象から除く）関数である。要素番号 pos＋1 以降の内容を一つずつ前の要素に移し，MemoCnt から1を引くことで，Memo[pos]が指すメモを削除する。Data[ ]中の参照されなくなったメモは，そのまま残すため，Data[ ]に対する処理はない。

行番号23：前に移動させる最初の要素の要素番号を設定
行番号24～27：Memo[pos ＋ 1]以降の要素を前から順に一つずつ前に移動
行番号28：現在格納されているメモの件数を一つ減らす

```
31  ○関数 changeMemo(整数型：pos, 整数型：textLen, 文字列型：text)
32    ○整数型：i
33    ・Memo[pos] ← DataLen
34    ・Data[DataLen] ← textLen
35    ・DataLen ← DataLen ＋ 1
36    ■ i：0, i ＜ textLen, 1
37    │ ・Data[DataLen ＋ i] ← text[i]
38    ■
39    ・DataLen ← [ b ]
```

changeMemo は，1件のメモの内容を変更する関数である。変更後の文字列を Data[ ]の最初の空き領域以降に格納し，その格納位置の情報を変更前の文字列の格納位置の情報と置き換えることでメモの内容を変更する。

行番号33：Data[ ]の最初の空き領域の要素番号を，変更するメモの位置情報として Memo[ ]に格納する
行番号34：Data[ ]の最初の空き領域に変更後の文字列の長さを格納
行番号35：Data[ ]に格納されている文字数を一つ増やす
　　これによって，DataLen は文字列 text の先頭文字の格納場所を表すことになる
行番号36～38：文字列 text を1文字ずつ Data[ ]に格納
行番号39：Data[ ]に格納されている文字数をメモの文字数分増やす

```
41  ○関数 moveMemo(整数型：fromPos, 整数型：toPos)
42    ○整数型：i, m
43    ・m ← Memo[fromPos]
44    ▲ fromPos ＜ toPos
45    │ ■ i：fromPos, i ≦ toPos － 1, 1
46    │ │ ・Memo[i] ← Memo[i ＋ 1]
47    │ ■
48    ▼
49    ▲ fromPos ＞ toPos
50    │ ■ i：[ d ]
51    │ │ ・Memo[i] ← Memo[i － 1]
52    │ ■
53    ▼
54    ・Memo[toPos] ← m
```

moveMemo は，1件のメモを移動する関数である。Memo[ ]の要素の並びを変えることでメモを移動する。

行番号43：移動元のメモの位置を退避
行番号44～48：移動元が移動先よりも前に存在する場合の処理
　　行番号45～47：移動元より後ろの要素を前から順に一つずつ前に移動
行番号49～53：移動元が移動先よりも後ろに存在する場合の処理
　　行番号50～52：移動元より前の要素を後ろから順に一つずつ後ろに移動
行番号54：退避した移動元のメモの位置を移動先に格納

［設問1］

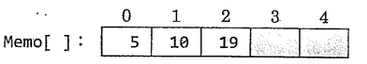
・空欄 a，b：関数 addMemo は，1件のメモを追加する処理を行う。空欄 a のある行番号3の処理は，Memo[MemoCnt]への値の格納である。Memo[ ]は，メモの格納位置の情報を管理する配列であり，Memo[MemoCnt]に入る値は，追加するメモの文字数を格納する Data[ ]の位置である。既に確認したように，追加するメモは，DataLen が指す位置から始まる空き要素に格納される。したがって，空欄 a には「DataLen」の（ア）が入る。

　続いて，空欄 b のある行番号10の処理は，メモ追加後の DataLen の値の更新である。DataLen は，行番号6で＋1 されるが，行番号7～9の文字列 text を Data[ ]に格納する処理では変化しない。そこで，空欄 b で text の文字数分を加算する処理が必要となる。文字列 text の文字数は，textLen に格納されているので，現在の DataLen の値に textLen を加えればよい。したがって，空欄 b には「DataLen ＋ textLen」の（ウ）が入る。

・空欄 c：関数 deleteMemo は，1件のメモを削除する処理を行う。メモの削除は，Memo[ ]の要素から削除するメモの格納位置の情報を削除し，Data[ ]を参照しないようにすることで行う。要素の削除は，削除する要素の次に続く要素を一つずつ前に移すことで行う。行番号25の処理が Memo[ ]の要素を一つ前に移す処理であるが，ここでは，移動元の要素番号を i，移動先の要素番号を i － 1 で表している。そのため，i は削除する要素の要素番号を表す pos の一つ後ろの要素番号（pos に＋1した値）である必要がある。したがって，「pos ＋ 1」の（エ）が入る。

・空欄 d：関数 moveMemo は，1件のメモを移動する処理を行う。メモの移動は，Memo[fromPos]を Memo[toPos]に移動させ，Memo[ ]の要素の並び順を変えることで行う。このとき，fromPos ＜ toPos の場合と fromPos ＞ toPos の場合とでは，Memo[ ]中の要素の移動方法が異なる。具体的には次のようになる。

【fromPos ＜ toPos の場合（moveMemo (0, 2) を実行)】



問8　簡易メモ帳のメモリ管理（データ構造及びアルゴリズム）（H28春-FE 午後問8）

【解答】

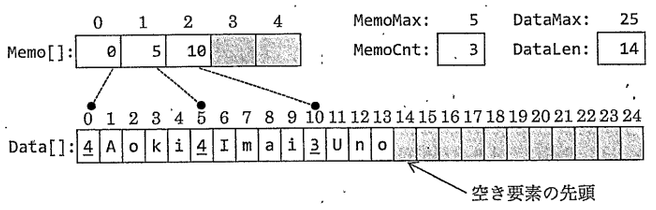
［設問1］　a－ア，b－ウ，c－エ，d－イ

［設問2］　e－ウ，f－カ，g－ケ

【解説】

簡易メモ帳のメモリ管理に関する問題である。難しそうなテーマであるが，内容は文字列の操作であり，配列の扱いを理解していれば対応できる。設問は，メモの編集処理（メモの追加・削除・変更・移動）を行うプログラムの空欄の穴埋めと参照されなくなったメモを取り除き，空き要素を増やすプログラムのトレース（追跡）の二つである。プログラムはいずれも短く，示された図で，配列を使ってどのようにデータを格納し，管理するのかを確認しながら処理をトレースすれば，確実に正解できる。

はじめに，メモの格納と管理をどのように行っているのかを確認する。メモの格納と管理には，2個の配列 Memo[ ], Data[ ]と4個の変数 MemoCnt, MemoMax, DataLen, DataMax を使用する。各メモは，メモの文字列の前に文字列の長さを付け加えて，Data[ ]の1要素に1バイトずつ格納する。そして，各メモの格納位置の情報（メモの文字長が格納されている Data[ ]の要素番号）を，メモの表示順に Memo[ ]の先頭の要素から順に設定して管理する。格納できるメモの最大件数（Memo[ ]の要素数）は MemoMax，現在格納されているメモの件数は MemoCnt に格納されている。また，メモを格納できる最大文字数（Data[ ]の要素数）は DataMax，Data[ ]に現在格納されている文字数は DataLen に格納されている。なお，DataLen は，Data[ ]に現在格納されている文字数を表すと同時に，Data[ ]の最初の空き要素の位置も表している。



メモの追加は，Data[ ]の最初の空き要素以降に格納する。メモの削除は，Data[ ]にはメモを残したまま，Memo[ ]から位置情報を削除することで行う。また，メモの変更は，変更前のメモは Data[ ]に残したまま，変更後のメモを Data[ ]の最初の空き要素以降に格納し，Memo[ ]の格納位置の情報を変更後の要素番号に変更することで行う。メモの移動は，Memo[ ]の要素の並び順を変えて，Memo[fromPos]の内容を Memo[toPos]の位置に移動する。このとき，まず Memo[fromPos]の要素を退避し，fromPos＜toPos の場合は，Memo[fromPos ＋ 1]～Memo[toPos]の要素を1要素ずつ前にずらし，fromPos＞toPos の場合は，Memo[fromPos － 1]から前方の Memo[toPos]までの要素を1要素ずつ後ろにずらしてから，退避した値を Memo[toPos]の位置に格納する。これらのことは，各関数の説明や関数を実行後の図から読み取ることができる。

続いて，各プログラムを確認する。

〔プログラム〕

○大域　整数型：MemoCnt, MemoMax, Memo[MemoMax]
○大域　整数型：DataLen, DataMax
○大域　8ビット論理型：Data[DataMax]

○関数：resetMemo()

・MemoCnt ← 0
・DataLen ← 0

resetMemo は，全てのメモを消去する関数である。MemoCnt と DataLen に 0 を設定することによって，Memo[ ]と Data[ ]の全要素を“空き”状態にする。

(行番号)

```
 1  ○関数：addMemo(整数型：textLen, 文字列型：text)
 2    ○整数型：i
 3    ・Memo[MemoCnt] ← [ a ]
 4    ・MemoCnt ← MemoCnt ＋ 1
 5    ・Data[DataLen] ← textLen
 6    ・DataLen ← DataLen ＋ 1
 7    ■ i：0, i ＜ textLen, 1
 8    │ ・Data[DataLen ＋ i] ← text[i]
 9    ■
10    ・DataLen ← [ b ]
```

addMemo は，1件のメモを追加する関数である。長さ textLen の文字列 text を Data[ ]の最初の空き要素以降に格納し，その格納位置の情報を Memo[ ]に設定する。

行番号3：追加したメモの格納位置の情報を Memo[ ]に設定
行番号4：メモの件数を一つ増やす
行番号5：Data[ ]の最初の空き領域に追加する文字列の長さを格納
行番号6：Data[ ]に格納されている文字数を一つ増やす
　　これによって，DataLen は文字列 text の先頭文字の格納場所を表すことになる
行番号7～9：文字列 text を1文字ずつ Data[ ]に格納
行番号10：Data[ ]に格納されている文字数をメモの文字数分増やす

```
21  ○関数：deleteMemo(整数型：pos)
22    ○整数型：i
23    ・i ← [ c ]
24    ■ i ＜ MemoCnt
25    │ ・Memo[i － 1] ← Memo[i]
26    │ ・i ← i ＋ 1
27    ■
28    ・MemoCnt ← MemoCnt － 1
```

deleteMemo は，1件のメモを削除する（表示の対象から除く）関数である。要素番号 pos＋1 以降の内容を一つずつ前の要素に移し，MemoCnt から1を引くことで，Memo[pos]が指すメモを削除する。Data[ ]中の参照されなくなったメモは，そのまま残すため，Data[ ]に対する処理はない。

行番号23：前に移動させる最初の要素の要素番号を設定
行番号24～27：Memo[pos ＋ 1]以降の要素を前から順に一つずつ前に移動
行番号28：現在格納されているメモの件数を一つ減らす

```
31  ○関数 changeMemo(整数型：pos, 整数型：textLen, 文字列型：text)
32    ○整数型：i
33    ・Memo[pos] ← DataLen
34    ・Data[DataLen] ← textLen
35    ・DataLen ← DataLen ＋ 1
36    ■ i：0, i ＜ textLen, 1
37    │ ・Data[DataLen ＋ i] ← text[i]
38    ■
39    ・DataLen ← [ b ]
```

changeMemo は，1件のメモの内容を変更する関数である。変更後の文字列を Data[ ]の最初の空き領域以降に格納し，その格納位置の情報を変更前の文字列の格納位置の情報と置き換えることでメモの内容を変更する。

行番号33：Data[ ]の最初の空き領域の要素番号を，変更するメモの位置情報として Memo[ ]に格納する
行番号34：Data[ ]の最初の空き領域に変更後の文字列の長さを格納
行番号35：Data[ ]に格納されている文字数を一つ増やす
　　これによって，DataLen は文字列 text の先頭文字の格納場所を表すことになる
行番号36～38：文字列 text を1文字ずつ Data[ ]に格納
行番号39：Data[ ]に格納されている文字数をメモの文字数分増やす

```
41  ○関数 moveMemo(整数型：fromPos, 整数型：toPos)
42    ○整数型：i, m
43    ・m ← Memo[fromPos]
44    ▲ fromPos ＜ toPos
45    │ ■ i：fromPos, i ≦ toPos － 1, 1
46    │ │ ・Memo[i] ← Memo[i ＋ 1]
47    │ ■
48    ▼
49    ▲ fromPos ＞ toPos
50    │ ■ i：[ d ]
51    │ │ ・Memo[i] ← Memo[i － 1]
52    │ ■
53    ▼
54    ・Memo[toPos] ← m
```

moveMemo は，1件のメモを移動する関数である。Memo[ ]の要素の並びを変えることでメモを移動する。

行番号43：移動元のメモの位置を退避
行番号44～48：移動元が移動先よりも前に存在する場合の処理
　　行番号45～47：移動元より後ろの要素を前から順に一つずつ前に移動
行番号49～53：移動元が移動先よりも後ろに存在する場合の処理
　　行番号50～52：移動元より前の要素を後ろから順に一つずつ後ろに移動
行番号54：退避した移動元のメモの位置を移動先に格納

［設問1］

・空欄 a，b：関数 addMemo は，1件のメモを追加する処理を行う。空欄 a のある行番号3の処理は，Memo[MemoCnt]への値の格納である。Memo[ ]は，メモの格納位置の情報を管理する配列であり，Memo[MemoCnt]に入る値は，追加するメモの文字数を格納する Data[ ]の位置である。既に確認したように，追加するメモは，DataLen が指す位置から始まる空き要素に格納される。したがって，空欄 a には「DataLen」の（ア）が入る。

　続いて，空欄 b のある行番号10の処理は，メモ追加後の DataLen の値の更新である。DataLen は，行番号6で＋1 されるが，行番号7～9の文字列 text を Data[ ]に格納する処理では変化しない。そこで，空欄 b で text の文字数分を加算する処理が必要となる。文字列 text の文字数は，textLen に格納されているので，現在の DataLen の値に textLen を加えればよい。したがって，空欄 b には「DataLen ＋ textLen」の（ウ）が入る。

・空欄 c：関数 deleteMemo は，1件のメモを削除する処理を行う。メモの削除は，Memo[ ]の要素から削除するメモの格納位置の情報を削除し，Data[ ]を参照しないようにすることで行う。要素の削除は，削除する要素の次に続く要素を一つずつ前に移すことで行う。行番号25の処理が Memo[ ]の要素を一つ前に移す処理であるが，ここでは，移動元の要素番号を i，移動先の要素番号を i － 1 で表している。そのため，i は削除する要素の要素番号を表す pos の一つ後ろの要素番号（pos に＋1した値）である必要がある。したがって，「pos ＋ 1」の（エ）が入る。

・空欄 d：関数 moveMemo は，1件のメモを移動する処理を行う。メモの移動は，Memo[fromPos]を Memo[toPos]に移動させ，Memo[ ]の要素の並び順を変えることで行う。このとき，fromPos ＜ toPos の場合と fromPos ＞ toPos の場合とでは，Memo[ ]中の要素の移動方法が異なる。具体的には次のようになる。

【fromPos ＜ toPos の場合（moveMemo (0, 2) を実行)】