1 アルゴリズムとプログラミング

問31 Check □ □ □ 【基本情報技術者試験 科目B試験サンプル問題 問3】

次のプログラム中の [a] と [b] に入れる正しい答えの組合せを，解答群の中から選べ。

手続appendは，引数で与えられた文字を単方向リストに追加する手続である。単方向リストの各要素は，クラスListElementを用いて表現する。クラスListElementの説明を図に示す。ListElement型の変数はクラスListElementのインスタンスの参照を格納するものとする。大域変数listHeadは，単方向リストの先頭の要素の参照を格納する。リストが空のときは，listHeadは未定義である。

メンバ変数	型	説明
val	文字型	リストに格納する文字。
next	ListElement	リストの次の文字を保持するインスタンスの参照。初期状態は未定義である。

コンストラクタ	説明
ListElement(文字型: qVal)	引数qValでメンバ変数valを初期化する。

図　クラスListElementの説明

〔プログラム〕

```
大域: ListElement: listHead ← 未定義の値

○append(文字型: qVal)
  ListElement: prev, curr
  curr ← ListElement(qVal)
  if (listHead が [ a ] )
    listHead ← curr
  else
    prev ← listHead
    while (prev.next が 未定義でない)
      prev ← prev.next
    endwhile
    prev.next ← [ b ]
  endif
```

解答群

	a	b
ア	未定義	curr
イ	未定義	curr.next
ウ	未定義	listHead
エ	未定義でない	curr
オ	未定義でない	curr.next
カ	未定義でない	listHead

1 アルゴリズムとプログラミング

問32 Check □ □ □ 【オリジナル問題】

次のプログラム中の [a] と [b] に入れる正しい答えの組合せを，解答群の中から選べ。ここで，配列の要素番号は０から始まる。

関数mergeは，整数が降順に格納された二つの整数型の配列arrayA，arrayBを引数として受け取り，配列arrayCに降順に併合（マージ）して返す。

関数mergeの実行例を，図１に示す。ここで，arrayCには，二つの配列を併合して格納できるだけの十分な要素数があるものとする。また，配列arrayA，arrayBに格納されている整数の並び順が正しくない場合などは考えなくてよい。

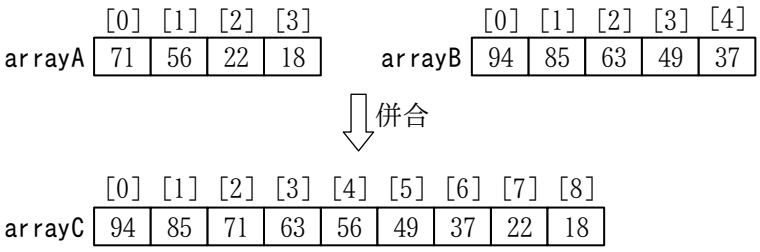

図１　関数mergeの実行例

〔プログラム〕

```
○整数型の配列: merge(整数型の配列: arrayA, 整数型の配列: arrayB)
  整数型の配列: arrayC
  整数型: numA, numB, idxA, idxB, idxC
  numA ← arrayAの要素数
  numB ← arrayBの要素数
  idxA ← 0
  idxB ← 0
  idxC ← 0
```

```
while (idxC が (numA ＋ numB) より小さい))
  if ((idxA が numA より小さい) and (idxB が numB より小さい))
    if ( [　　a　　] 大きい)
      arrayC[idxC] ← arrayA[idxA]
      idxA ← idxA ＋ 1
    else
      arrayC[idxC] ← arrayB[idxB]
      idxB ← idxB ＋ 1
    endif
  elseif ( [　　b　　] 小さい)
      arrayC[idxC] ← arrayA[idxA]
      idxA ← idxA ＋ 1
  else
      arrayC[idxC] ← arrayB[idxB]
      idxB ← idxB ＋ 1
  endif
  idxC ← idxC ＋ 1
endwhile
return arrayC
```

解答群

	a	b
ア	arrayA[idxA] が arrayB[idxB] より	idxA が idxB より
イ	arrayA[idxA] が arrayB[idxB] より	idxA が numA より
ウ	arrayA[idxA] が arrayB[idxB] より	idxB が idxA より
エ	arrayA[idxA] が arrayB[idxB] より	idxB が numB より
オ	arrayB[idxB] が arrayA[idxA] より	idxA が idxB より
カ	arrayB[idxB] が arrayA[idxA] より	idxA が numA より
キ	arrayB[idxB] が arrayA[idxA] より	idxB が idxA より
ク	arrayB[idxB] が arrayA[idxA] より	idxB が numB より

1 アルゴリズムとプログラミング

問33 Check □ □ □ 【オリジナル問題】

次のプログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。ここで，配列の要素番号は1から始まる。

関数sentinelLoopは，数値が格納された整数型の配列arrayと，格納されている数値の個数nを引数として受け取り，同じく引数として受け取った整数型の変数numと同じ数値が格納された要素array[i]の要素番号iを返す。このとき，変数numと同じ数値が配列array中に存在しない場合は，（－1）を返す。

なお，配列arrayの要素array[n＋1]は，作業用として使用できるものとする。

〔プログラム〕

```
○整数型: sentinelLoop(整数型の配列: array, 整数型: n, 整数型: num)
  整数型: i
  array[n ＋ 1] ← num
  i ← 1
  while ( [                    ] )
    i ← i ＋ 1
  endwhile
  if (i が n 以下)
    return i
  else
    return (－1)
  endif
```

解答群

ア　(array[i] が num と等しい) and (i が n 以下)

イ　(array[i] が num と等しい) or (i が n より小さい)

ウ　(array[i] が num と等しくない) and (i が n より小さい)

エ　(array[i] が num と等しくない) or (i が n 以下)

オ　array[i] が num と等しい

カ　array[i] が num と等しくない

キ　i が n 以下

ク　i が n より小さい

1 アルゴリズムとプログラミング

問34 Check □ □ □ 【オリジナル問題】

次のプログラム中の [a] と [b] に入れる正しい答えの組合せを，解答群の中から選べ。

手続insertElementは，引数で与えられた整数を，単方向リストの並び順を崩さないように追加する手続である。

単方向リストの各要素はクラスElementを用いて表現し，単方向リストはメンバ変数valの昇順に並べられている。クラスElementの説明を，図1に示す。Element型の変数は，クラスElementのインスタンスの参照を格納するものとする。大域変数topは，単方向リストの先頭の要素の参照を格納する。リストが空のとき，topは未定義である。

手続insertElementは，引数numに受け取った整数をメンバ変数valとするクラスElementのインスタンスを生成し，単方向リストの適切な位置に追加する。なお，引数numと同じ値のメンバ変数valをもつ要素があった場合，その要素の前の位置に追加するものとする。

メンバ変数	型	説明
val	整数型	リストに格納する整数。
next	Element	リストの次の整数を保持するインスタンスの参照。初期状態は未定義である。

コンストラクタ	説明
Element(整数型: qVal, Element: qNext)	引数qValでメンバ変数valを，引数qNextでメンバ変数nextを初期化する。

図1　クラスElementの説明

〔プログラム〕

```
大域: Element: top ← 未定義の値

○insertElement(整数型: num)
  Element: prev, curr, work
  curr ← top
  while ((curr が 未定義でない) and ( [ a ] ))
    prev ← curr
    curr ← curr.next
  endwhile
  work ← [ b ]
  if (curr が top と等しい)
    top ← work
  else
    prev.next ← work
  endif
```

解答群

	a	b
ア	curr.val が num より大きい	Element(num, curr)
イ	curr.val が num より大きい	Element(num, curr.next)
ウ	curr.val が num より大きい	Element(num, prev)
エ	curr.val が num より小さい	Element(num, curr)
オ	curr.val が num より小さい	Element(num, curr.next)
カ	curr.val が num より小さい	Element(num, prev)

1 アルゴリズムとプログラミング

問35 Check □ □ □ 【オリジナル問題】

次のプログラム中の［ a ］と［ b ］に入れる正しい答えの組合せを，解答群の中から選べ。ここで，配列の要素番号は０から始まる。

配列によるデータ管理方法の一つにハッシュテーブルがある。要素のキー値から，一定の規則（ハッシュ関数）によって要素の格納位置を求め，ハッシュテーブルにデータを追加するプログラムである。

関数hashAddは，引数で与えられたデータをハッシュテーブルに追加し，データの格納位置（要素番号）を返す。ハッシュテーブルに格納するデータは，クラスHashElemを用いて表現する。クラスHashElemの説明を，図１に示す。大域変数として定義されたHashElem型の配列htの要素には，データを表すクラスHashElemのインスタンスの参照が格納されている。また，配列htの要素が未使用の場合，その要素は**未定義**である。

メンバ変数	型	説明
hKey	整数型	データのキー値
hDat	整数型	データの値

図１　クラスHashElemの説明

関数hashAddは，追加するデータのキー値（正の整数）を配列htの要素数で除算したときの剰余を，格納位置として使用する。ただし，キー値から求めた格納位置の要素がすでに使用されている場合，後続の未使用の要素に追加する。このとき，配列の要素は環状に連続しているものとして処理する。

関数hashAddの実行例を，図２に示す。273を要素数（５）で除算した剰余は３になるので，“① ht[3] → ② ht[4] → ③ ht[0] → ④ ht[1]”の順に未使用の要素を探し，最終的にht[1]に追加する。なお，配列htに未使用の要素がない場合は，データを追加せずに（－1）を返す。

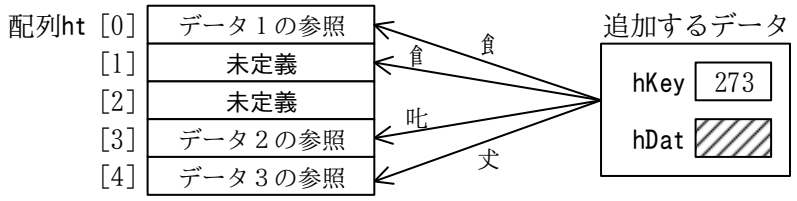

図２　関数hashAddの実行例

〔プログラム〕

```
大域: HashElemの配列: ht          /* ハッシュテーブル */
大域: 整数型: n                   /* ハッシュテーブルの要素数 */

○整数型: hashAdd(HashElem: parm)
  整数型: hash, i, pos
  hash ← parm.hKey mod n
  if (ht[hash] が 未定義)
    pos ← hash
  else
    pos ← (−1)
    i ← hash + 1
    while (( [ a ] ) and (pos が (−1) と等しい))
      if ( [ b ] )
        i ← 0
      endif
      if (ht[i] が 未定義)
        pos ← i
      endif
      i ← i + 1
    endwhile
  endif
  if (pos が (−1) と等しくない)
    ht[pos] ← parm
  endif
  return pos
```

解答群

	a	b
ア	ht[i] が 未定義でない	(hash + i) が n と等しい
イ	ht[i] が 未定義でない	ht[i] が parm と等しい
ウ	ht[i] が 未定義でない	i が n と等しい
エ	i が hash と等しくない	(hash + i) が n と等しい
オ	i が hash と等しくない	ht[i] が parm と等しい
カ	i が hash と等しくない	i が n と等しい

1 アルゴリズムとプログラミング

問36 Check □ □ □ 【オリジナル問題】

次のプログラム中の [a] と [b] に入れる正しい答えの組合せを，解答群の中から選べ。ここで，配列の要素番号は０から始まる。

関数gradeChangeは，試験の得点scoreを引数として受け取り，その得点に該当する成績（Ａ～Ｄ）を返す。

この試験は100点満点で，成績の判定基準は表１のとおりである。なお，引数で受け取った得点scoreの値に誤りがある場合は，エラーとして“Ｘ”を返す。

表１　成績の判定基準

得点の範囲	80～100点	60～79点	40～59点	0～39点
成績	A	B	C	D

〔プログラム〕

```
○文字型: gradeChange(整数型: score)
  整数型の配列: scoreRange ← {100, 79, 59, 39, −1}
  文字型の配列: grade ← {'A', 'B', 'C', 'D', 'X'}
  整数型: i, j, k
  if ((score が 0 未満) or (score が 100 より大きい))
    return grade[4]
  else
    i ← 0
    j ← scoreRangeの要素数 − 1
    while (i が j 以下)
      k ← (i + j) ÷ 2
      if ((score が scoreRange[k] 以下) and (score が [ a ]))
        return grade[k]
      elseif (score が [ b ] )
        j ← k − 1
      else
        i ← k + 1
      endif
    endwhile
    return grade[4]
  endif
```

解答群

	a	b
ア	scoreRange[k ＋ 1] 以上	scoreRange[k ＋ 1] より小さい
イ	scoreRange[k ＋ 1] 以上	scoreRange[k － 1] より大きい
ウ	scoreRange[k ＋ 1] 以上	scoreRange[k] より大きい
エ	scoreRange[k ＋ 1] 以上	scoreRange[k] より小さい
オ	scoreRange[k ＋ 1] より大きい	scoreRange[k ＋ 1] より小さい
カ	scoreRange[k ＋ 1] より大きい	scoreRange[k － 1] より大きい
キ	scoreRange[k ＋ 1] より大きい	scoreRange[k] より大きい
ク	scoreRange[k ＋ 1] より大きい	scoreRange[k] より小さい