CHAPTER

2

觀念題如何考 4 級分

2.6 APCS 觀念題實例演習

題號 1

- **使用哪些概念或技巧：陣列、存取範圍**
- **出題者想要考考生的考點：陣列的邊界檢查、陣列索引**

給定程式片段，哪個 n 值不會造成超過陣列 A 的存取範圍？

```
1  int i, n, A[100];
2  scanf ("%d", &n);
3  for (i=0; i!=n; i=i+1) {
4      A[i] = i;
5      i = i + 1;
6  }
```

(A) 69　(B) 89　(C) 98　(D) 202

解析

初始化變數 i 和 n 以及陣列 A，陣列 A 的大小是 100。

1. **在每次迴圈中**：

   A[i] = i 將 i 的值賦給 A[i]。

   i = i + 1 再次將 i 增加 1。

   注意到 i 在每次迴圈中實際上增加了兩次（在 for 的增量部分和在迴圈體內），所以每次迴圈實際上 i 會增加 2。

2. **讓我們列出每個 i 的值變化範圍**：

   i 的值變化為：0, 2, 4, 6, 8, ..., 2k

條件 i != n 意味著當 i 等於 n 時迴圈會結束，所以 i 的最大值應該小於 100（因為陣列 A 的大小是 100）。由於每次 i 增加 2，i 值只會是偶數，i 的最大值應該不超過 98。

3. **現在我們來檢查選項**：

(A) 69

非偶數。

(B) 89

非偶數。

(C) 98

當 n = 98 時，i 最多能夠達到 98，不會超過陣列範圍。

(D) 202

當 n = 202 時，i 會超過 100，會超出陣列範圍。

所以答案不是 (A), (B) 和 (D)

答案：(C) 98

題號 2

使用哪些概念或技巧：函數、遞迴
出題者想要考考生的考點：遞迴函數的運作

給定函式 f( )，當執行 f(10) 時，最終回傳結果為何？

```
int f (int i) {
    if (i > 0) {
        if (((i / 2) % 2) == 0)
            return f(i - 2) * i;
        else
            return f(i - 2) * (-i);
    } else {
        return 1;
```

```
9         }
10  }
```

(A) 1　(B) 3840　(C) -3840　(D) 執行時導致無窮迴圈，不會停止執行

解析

分析一下這個函式的遞迴過程：

1. **當 i 大於 0 時**：
   - 如果 (i / 2) % 2 == 0，即 i 除以 2 的結果是偶數，則回傳 f(i - 2) * i。
   - 否則回傳 f(i - 2) * (-i)。
2. **當 i 小於或等於 0 時，回傳 1。**

接著，我們逐步計算 f(10)：

- f(10)：
  - 10 / 2 = 5，5 % 2 = 1，不是偶數。
  - 回傳 f(8) * (-10)。
- f(8)：
  - 8 / 2 = 4，4 % 2 = 0，是偶數。
  - 回傳 f(6) * 8。
- f(6)：
  - 6 / 2 = 3，3 % 2 = 1，不是偶數。
  - 回傳 f(4) * (-6)。
- f(4)：
  - 4 / 2 = 2，2 % 2 = 0，是偶數。
  - 回傳 f(2) * 4。

- f(2)：
  - 2 / 2 = 1，1 % 2 = 1，不是偶數。
  - 回傳 f(0) * (-2)。
- f(0)：
  - i <= 0，回傳 1。

然後我們將這些結果連續代入：

- f(0) = 1
- f(2) = f(0) * (-2) = 1 * (-2) = -2
- f(4) = f(2) * 4 = -2 * 4 = -8
- f(6) = f(4) * (-6) = -8 * (-6) = 48
- f(8) = f(6) * 8 = 48 * 8 = 384
- f(10) = f(8) * (-10) = 384 * (-10) = -3840

因此，f(10) 的最終回傳結果是 -3840。

答案：(C) -3840

題號 3

使用哪些概念或技巧：for 迴圈、次數計算
出題者想要考考生的考點：迴圈執行次數的計算、非固定值的迴路增值

給定程式片段，for 迴圈總共會執行幾次？

```
1  int i, j = 0;
2  for (i = 0; i < 128; i = i + j) {
3      j = i + 1;
4  }
```

(A) 8　(B) 32　(C) 64　(D) 128

解析

我們可以逐步計算每次迴圈迭代時 i 和 j 的值變化。

1. **初始化**：

   i = 0

   j = 0

2. **進入迴圈**：

   - 第 1 次迭代：i = 0
     - j = i + 1 = 0 + 1 = 1
     - i = i + j = 0 + 1 = 1
   - 第 2 次迭代：i = 1
     - j = i + 1 = 1 + 1 = 2
     - i = i + j = 1 + 2 = 3
   - 第 3 次迭代：i = 3
     - j = i + 1 = 3 + 1 = 4
     - i = i + j = 3 + 4 = 7
   - 第 4 次迭代：i = 7
     - j = i + 1 = 7 + 1 = 8
     - i = i + j = 7 + 8 = 15
   - 第 5 次迭代：i = 15
     - j = i + 1 = 15 + 1 = 16
     - i = i + j = 15 + 16 = 31

- 第 6 次迭代：i = 31
  - j = i + 1 = 31 + 1 = 32
  - i = i + j = 31 + 32 = 63
- 第 7 次迭代：i = 63
  - j = i + 1 = 63 + 1 = 64
  - i = i + j = 63 + 64 = 127
- 第 8 次迭代：i = 127
  - j = i + 1 = 127 + 1 = 128
  - i = i + j = 127 + 128 = 255

在第 8 次迭代後，i 的值變為 255，這超過了 128，迴圈終止。因此，迴圈總共執行了 8 次。

答案：(A) 8

題號 4

使用哪些概念或技巧：程式輸出、條件判斷
出題者想要考考生的考點：條件判斷的理解

給定程式，若已知輸出的結果為 [1][2][3][5][4][6]，程式中的 (?) 應為下列何者？

```
1  int main() {
2      int i, j;
3      for (i = 0; i < 5; i = i + 1) {
4          for (j = 0; (?) ; j = j + 2) {
5              printf("[%d]", i + j);
6          }
7      }
8  }
```

(A) j < i　(B) j > i　(C) j <= i　(D) j >= i

解析

1. **觀察輸出結果 [1][2][3][5][4][6]**：
   - 輸出的數字規律：第一層迴圈 (i) 從 0 到 4，每次 (i) 都增加。
   - 第二層迴圈 (j) 從 0 開始，增加 2，直到不符合條件為止。
2. **分析條件**：
   - (i=0) 時，(j=0)，(j=2)：輸出 [1]
   - (i=1) 時，(j=1)，(j=3)：輸出 [2][3]
   - (i=2) 時，(j=2)，(j=4)：輸出 [5]
   - (i=3) 時，(j=3)：輸出 [4]
   - (i=4) 時，(j=4)：輸出 [6]
3. **分析條件 (?)**：
   - 要符合所有的情況，當 (j) 不再小於 (i) 時停止，所以條件應該是 j < i。

結論：根據上述分析，正確答案為 j < i。

答案：(A) j < i

CHAPTER

6

挑戰 APCS 8 級分以上

也就是目標是觀念 4 級，實作 4 級，以下是要再加強的部分。

6.1 貪心演算法

貪心演算法（Greedy Algorithm）是一種在每一步選擇中都採取當前最佳或最有利選擇的方法。這種方法希望透過一系列局部最優的選擇來達到全局最優的結果。貪心演算法的基本思路是，每次做出當前狀態下的最優選擇，而不考慮未來的結果。這種策略通常能夠在許多問題中得到快速而有效的解決方案，但它並不總能保證全局最優解。

貪心演算法的基本特徵

- **局部最優選擇**：在每一步中，選擇當前看起來最好的選擇。
- **無後效性**：一旦做出某個選擇，它不會再被改變或撤回。
- **可行性**：每一步所做的選擇都是當前可行的。
- **最終性**：所有的局部最優選擇加起來構成了問題的解。

貪心演算法的應用

1. **活動選擇問題**
    - 給定一組活動，每個活動都有一個開始時間和結束時間，要求選出最多數量的互不重疊的活動。貪心策略是每次選擇結束時間最早的活動。
2. **最小生成樹**
    - 在圖論中，最小生成樹問題可以使用貪心演算法來解決，例如 Kruskal 和 Prim 算法。

3. **霍夫曼編碼**
    - 霍夫曼編碼是一種無損數據壓縮演算法，它使用貪心策略來構造最優二叉樹。
4. **硬幣找零問題**
    - 在找零問題中，給定不同面值的硬幣以及總額，要求用最少數量的硬幣來湊成總額。貪心策略是每次選擇面值最大的硬幣。

貪心演算法的設計步驟

1. **選擇貪心策略**：確定如何在每一步選擇當前最優解。
2. **證明貪心選擇性**：證明局部最優選擇能導致全局最優解。
3. **構造貪心算法**：實作貪心策略的算法。
4. **分析時間複雜度**：確定算法的時間和空間複雜度。

範例 1：貪心演算法解決硬幣找零問題

假設我們有不同面額的硬幣，並且希望找零時使用最少數量的硬幣。

```
# 可用的硬幣面額
coins = [25, 10, 5, 1]
# 需要找零的金額
amount = 87

# 儲存找零硬幣的結果
change = []
remaining_amount = amount

for coin in coins:
    while remaining_amount >= coin:
        change.append(coin)
        remaining_amount -= coin
```

```
14
15  print(f"找零的硬幣： {change}")
16  print(f"總共使用了 {len(change)} 個硬幣")
```

在這段程式碼中，我們先定義了可用的硬幣面額 coins 列表，然後設定需要找零的金額 amount。接著，我們使用貪心演算法來找零，優先使用面額最大的硬幣，直到找零完成。

執行結果

```
找零的硬幣： [25, 25, 25, 10, 1, 1]
總共使用了 6 個硬幣
```

範例 2：貪心演算法解決活動選擇問題

```
1   活動選擇問題是一個典型的貪心演算法問題，我們需要選擇最多數量的互不重疊的活動。假
    設活動按照結束時間已經排序。
2   # 活動的開始和結束時間
3   activities = [
4       (1, 4),  # 活動 1: 開始時間 1, 結束時間 4
5       (3, 5),  # 活動 2: 開始時間 3, 結束時間 5
6       (0, 6),  # 活動 3: 開始時間 0, 結束時間 6
7       (5, 7),  # 活動 4: 開始時間 5, 結束時間 7
8       (3, 8),  # 活動 5: 開始時間 3, 結束時間 8
9       (5, 9),  # 活動 6: 開始時間 5, 結束時間 9
10      (6, 10), # 活動 7: 開始時間 6, 結束時間 10
11      (8, 11), # 活動 8: 開始時間 8, 結束時間 11
12      (8, 12), # 活動 9: 開始時間 8, 結束時間 12
13      (2, 13), # 活動 10: 開始時間 2, 結束時間 13
14      (12, 14)# 活動 11: 開始時間 12, 結束時間 14
15  ]
16
17  # 按結束時間排序活動
18  activities.sort(key=lambda x: x[1])
19
20  # 儲存選擇的活動
```

```
selected_activities = []
# 上一個選擇的活動結束時間
last_end_time = 0

for activity in activities:
    start, end = activity
    if start >= last_end_time:
        selected_activities.append(activity)
        last_end_time = end

print(f"選擇的活動：{selected_activities}")
```

這段程式碼執行了以下步驟：

1. 定義活動的開始和結束時間。
2. 按照活動的結束時間對活動進行排序。
3. 使用貪心演算法選擇最多數量的互不重疊的活動。

執行結果將會顯示選擇的活動：

```
選擇的活動：[(1, 4), (5, 7), (8, 11), (12, 14)]
```

貪心演算法是一種簡單而有效的策略，適用於許多問題，但需要小心驗證它能否得到全局最優解。透過理解和應用貪心策略，可以在 APCS 考試中輕鬆應對相關題型，並取得好成績。

6.2 分治演算法

分治演算法（Divide and Conquer）是一種將原問題分解為若干個規模較小的子問題，遞迴地解決這些子問題，然後再合併子問題的解來得到原問題的解的方法。分治法通常適用於問題規模較大且結構比較規則的情況。

分治演算法的基本步驟

1. **分解（Divide）**：將問題分解為若干個規模較小的子問題。
2. **解決（Conquer）**：遞迴地解決這些子問題。如果子問題規模足夠小，則直接解決。
3. **合併（Combine）**：將子問題的解合併成原問題的解。

分治演算法的應用

1. **合併排序（Merge Sort）**
   - 合併排序是一種典型的分治演算法，它將數列不斷地分成兩半，對每個子序列進行排序，然後再合併已排序的子序列。
2. **快速排序（Quick Sort）**
   - 快速排序也是一種典型的分治演算法。它選擇一個基準元素，將數列分成兩部分，一部分所有元素都小於基準元素，另一部分所有元素都大於基準元素，然後對這兩部分分別遞迴排序。
3. **大數乘法（Karatsuba Algorithm）**
   - Karatsuba 算法是一種高效的大數乘法演算法，利用分治法將大數的乘法轉化為若干個較小數的乘法和加法。
   - 例子：合併排序

合併排序的步驟

1. **分解**：將數列分成兩半。
2. **解決**：對每一半數列進行遞迴合併排序。
3. **合併**：將兩個已排序的子序列合併成一個有序的序列。

範例：合併排序的 Python 實做

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        # 遞迴分治
        merge_sort(left_half)
        merge_sort(right_half)

        # 合併
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1

def main():
    arr = [12, 11, 13, 5, 6, 7]
    print("Given array is:", arr)
```

```
34      merge_sort(arr)
35      print("Sorted array is:", arr)
36
37  if __name__ == "__main__":
38      main()
```

解析

1. **merge_sort 函數**：
   - 如果數列長度大於 1，則將數列分成兩半，分別進行合併排序。
   - 合併兩個已排序的子序列。
2. **main 函數**：
   - 定義一個待排序的數列，調用 merge_sort 函數進行排序，並印出排序前後的數列。

合併排序的時間複雜度

合併排序的時間複雜度是 O(n log n)，因為它將數列不斷地分成兩半（log n 次），並且每次合併需要 O(n) 的時間。

分治法的優點

- **效率高**：對於很多問題，分治法能顯著降低時間複雜度。
- **易於並行**：因為每個子問題可以獨立解決，分治法特別適合於並行計算。

分治演算法是一種強大的問題解決策略，適用於許多複雜的問題。透過理解和應用分治策略，可以在 APCS 考試中輕鬆應對相關題型，並取得好成績。