1131 NTOUCSE 程式設計 1C 期末考參考答案

姓名：_____ 系級：_____ 學號：_____ **113/12/24** (二)

考試時間：**13:20 – 16:00**

考試規則：1. **請闔上課本**，**不可**參考任何文件包括小考、作業、實習、或是其它資料
2. 你可以在題目卷上直接回答，可以使用鉛筆，但是**請在答案卷上寫下題號，並且註明答案在題目卷上**
3. 有需要的話，可以使用沒有教過的語法，但是**僅限於 C 語言**
4. 程式撰寫時請寫完整的程式碼，寫... 的分數很低 (程式裡有很多重複的敘述本來就是扣分的)
5. **不可**使用電腦、平板、電子紙、智慧手機、手錶、及工程型計算機
6. 請**不要**左顧右盼！請勿討論！請勿交換任何資料！對於題目有任何疑問請舉手發問
7. 如果你提早交卷，請迅速安靜地離開教室，請勿在走廊喧嘩
8. 違反上述考試規則視為不誠實的行為，由學校依學務規章處理
9. 請在**題目卷**及**答案卷**上都**寫下姓名及學號**，交卷時請繳交**題目卷**及**答案卷**

1. 在 C/C++ 語言中基礎的字串是內容以 '\0' 結束的字元陣列，請回答下列程式問題：

(a) [3] 請問以 char buf[10]; 定義的陣列本來就可以存放 ASCII/Big5/Utf-8/Utf-16 編碼的 10 個位元組的英文字元或是多位元組字元，為什麼需要在實際上存放的字串結束時放一個字元？請問結束字元是什麼？

Sol:

The terminating character of every character string is **'\0'** (ASCII encoding is 0). It is perfectly correct to store 10 arbitrary characters in this **char** array if your purpose is to store these data only and not using any C string library functions in stdio.h or string.h. If you want to **use C utility functions** like scanf("%s", puts(), fgets(), strlen(), strcmp(), strcat(), strcpy(), strtok(), …, these functions assume that each character string ends with the '\0' (or simply integer 0) such that a loop like

```
01 int i = 0;
02 while (buf[i]!=0)
03    some processing
```

can stop processing at the end of the string.

(b) [4] 請問下列定義 str1 與 str2 的方法在第 03 列和第 04 列印出來的字串有什麼不同？請舉例說明這兩個定義對於什麼樣的使用方法會有差異？

```
01 const char *str1="Hello World!";
02 char str2[]="Hello World!";
03 printf("%s", str1);
04 printf("%s", str2);
```

Sol:

The outputs from line (3) and line (4) are **exactly the same**. You can use either one of them in your program whenever the syntax requires a char * **unless you want to modify the content of the string**. In that case, **str1** points to an immutable character array with "Hello World!" prestored in it while **str2** points to 13 bytes of contiguous memory for your program to access or modify. For example, str2[6] = 'w'; is correct while str1[6] = 'w'; is incorrect. strcpy(str2, "Happy Hour!"); is correct while strcpy(str1, "Happy Hour!"); is not.

(c) [4] 請問一個字串常數 "Hello World!" 除了在題 (b) 第 02 列的用法之外對編譯器來說是什麼型態？請由語法解釋 printf("%c", *("Hello World!"+4)); 為什麼會印出 o？

Sol:

It is a character array with immutable contents which occupies 13-byte of contiguous memory. To the compiler, it is a const char array without a name. Its type is the same as **str** - the name of a const char array in the following definition:
const char str[] = { 'H','e','l','l','o',' ','W','o','r','l','d','!','\0'};
i.e. its type is **const char *const**.

Now that "Hello World!" is a char pointer, the arithmetic pointer expression +4 would add 4 times the size of a character, i.e. sizeof(char) to that pointer. The * is the dereference operator and obtains the content of the memory at the address "Hello World!"+4, i.e. the 5-th element 'o' of this constant character array. Note that this indirect access of memory *("Hello World!"+4) through an address is the same as "Hello World!"[4] or const char *str="Hello World!"; followed by str[4].

(d) [8] 下面程式第 05,06,07 列後面的註釋是某一次執行時印出的結果，請根據語法解釋第 05,06,07 列印出來的結果? (請解釋這幾個數值的意義)，如果第 07 列改為 printf("%p\n",buf+10); 請問列印出來的結果為何? 請問第 08 列執行以後鍵盤輸入的資料會寫到哪裡? 最多能夠讀幾個字元不會出現記憶體存取的錯誤? 第 09 列執行以後 scanf() 函式會把資料寫到哪裡? 會出現錯誤嗎?

```
01 #include <stdio.h>
02 int main()
03 {
04        char buf[]="SweetHome";
05        printf("%p\n", buf);       // 0000000000CFF938
06        printf("%p\n", buf+1);     // 0000000000CFF939
07        printf("%p\n", &buf+1); // 0000000000CFF942
08        scanf("%s", buf+1);
09        scanf("%s", &buf+1);
10        return 0;
11 }
```

Sol:

The format conversion command **%p** of printf() outputs the memory address of the corresponding argument in hexadecimal. At line 5, **buf** is the address of the first element of the 10-char array, whose type is **char*** and has the value $(0000000000CFF938)_{16}$ in a particular run. At line 6, the plus sign in the expression **buf+1** is a pointer addition, and the address **buf** is added by an offset of storage for 1 **char**, i.e. sizeof(**char**), and is $(0000000000CFF938)_{16}+1 = (0000000000CFF939)_{16}$. At line 7, **&buf** has type **char (*)[10]**, thus this address is incremented by an offset of storage for 1 **char[10]**, and is $(0000000000CFF938)_{16}+10 = (0000000000CFF942)_{16}$.

If line 7 is modified to printf("%p\n", **buf+10**);, the address written out on the screen will still be 0000000000CFF942 = $(0000000000CFF938)_{16}+10*1$, which is **buf** +10*sizeof(**char**).

At line 8, scanf() will read upto **8** characters and store those characters including one extra terminating character **'\0'** to memory starting from address **buf+1** (which is $(0000000000CFF939)_{16}$) to address **buf+9** (which is $(0000000000CFF941)_{16}$) without getting any runtime memory access error. Note that the data in the address **buf** , which is 'S', will not be touched by the scanf() function call in line 8.

In line 9, the statement tells scanf() to store whatever characters read from the keyboard to memory starting at $(0000000000CFF942)_{16}$, this memory is outside the allocated memory for the character array **buf[10]** and is not allowed to access by this program. Once scanf() tries to write data into the memory at address **&buf+1** the system is likely to report memory access error and aborts the execution of the program.

2. 請完成下列迴圈以及遞迴兩個版本的列舉 C(n,k) 程式

(a) [6] 迴圈版本在一個可以存放 k 位數的整數陣列 x[] 中數數字，例如右圖中列舉由 {1,2,3,4,5} 中任取 3 個元素時 C(5,3)=10 種可能的組合，請完成 07-13 列進位的規則，第 07 列實作第 i 位數的上限，第 10,11 列在第 i 位加 1 以後設定第 i+1 位到第 k-1 位的起始數值

```
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
```

```
01 #include <stdio.h>
02
03 int next(int x[], int n, int k)
04 {
```

```
05      int i, j;
06      for (i=k-1; i>=0; i--)
07          if (x[i] < _____________________)
08          {
09              x[i]++;
10              for (j=_____________; j<_____________; j++)
11                  x[j] = _________________
12              return 1;
13          }
14      return 0;
15 }
16
17 int main()
18 {
19      int i, x[10], n, k;
20      scanf("%d%d", &n, &k);
21      for (i=0; i<k; i++)
22          x[i] = i+1;
23      do
24          for (i=0; i<k; i++)
25              printf("%d%c", x[i], "\n "[i<k-1]);
26      while (next(x, n, k));
27      return 0;
28 }
```

Sol:

```
07          if (x[i] <  n-(k-1-i)  )
08          {
09              x[i]++;
10              for (j=  i+1  ; j<  k  ; j++)
11                  x[j] =  x[j-1]+1 ;
12              return 1;
13          }
```

(b) [10] 這個程式的遞迴版本比迴圈版本簡短，邏輯比較清晰，實現概念上可變層數的多層 for 迴圈，是一個基本的深度優先搜尋(Depth First Search, DFS)，撰寫時需要知道決策樹的樹狀架構，遞迴模板中包含運用遞迴函式參數定義的狀態、結束條件檢查、以及多種選項遞迴展開三部份，遞迴狀態參數是目前填寫陣列的索引 p，請完成下列第 02 列的定義，第 04 列結束條件的檢查，第 08 列 x[p] 所有選項的列舉，以及第 09 列遞迴進行下一層的搜索，另外請問第 15 列第三個參數 d+1 語法上的效果以及邏輯上的原因?

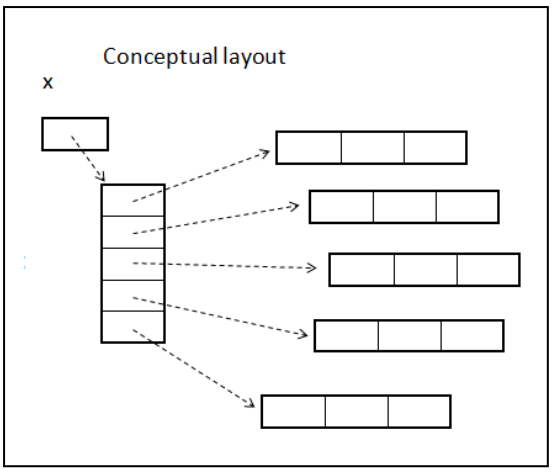
```
01 #include <stdio.h>
02 void comb(int n, int k, int x[], _____________)
03 {
04      if (_____________)
05          for (int i=0; i<k; i++)
06              printf("%d%c", x[i], "\n "[i<k-1]);
07      else
08          for (x[p]=_____________; x[p]<_____________; x[p]++)
09              comb(n, k, x, __________);
10 }
11 int main()
12 {
13      int n, k, d[10]={0};
14      scanf("%d%d", &n, &k);
15      comb(n,k,d+1,0);
16      return 0;
17 }
```

Sol:

```
02 void comb(int n, int k, int x[],___int p___)
03 {
04      if (___p>=k___) // or if (___p==k___)
05          for (int i=0; i<k; i++)
06              printf("%d%c", x[i], "\n "[i<k-1]);
07      else
08          for (x[p]=___x[p-1]+1___; x[p]<___n-(k-p-2)___; x[p]++)
09              comb(n, k, x,___p+1___);
10 }
```

The third argument **d+1** of the recursive function call at line 15 is the address of the second element of array d[], i.e. **&d[1]**. In this way, we want to map the subarray **d[1], d[2], …, d[9]** in main() to the array **x[0], x[1], …, x[8]** in the recursive function comb() and furthermore map **d[0]** in main, which is initialized to **0**, to **x[-1]** in comb(). The logical reason lies in the expression **x[p]=x[p-1]+1** at line 08, when p is 0, this expression uses **x[-1]** as its previous element which is required to be a constant **0**.

3. (a) [9] 請完成下列程式片段以右圖的方式動態配置一個二維的整數陣列 x，用來存放下列串流裡 m 列 n 行的整數矩陣 (第一列是 m 及 n，接下來有 m 列，每列 n 個整數)，這個陣列在第 28 列傳入 release() 函式以後，函式裡會釋放掉所配置的記憶體。在第 14 列請定義變數 x，第 18 列請運用 stdlib.h 裡面定義的 malloc() 函式配置右圖中間的指標陣列，變數所需要的位元組數請使用 sizeof() 來取得，第 20 列在迴圈中請再運用 malloc() 函式配置圖中右側五塊獨立的記憶體，第 24 列請以 scanf() 工具由串流中將資料讀入陣列 x，第 28 列呼叫函式 release(x, m) 釋放掉所配置的記憶體，第 4 列請定義 release() 函式的參數 x，第 07,08 列請以 free() 函式完全釋放掉所配置的記憶體。



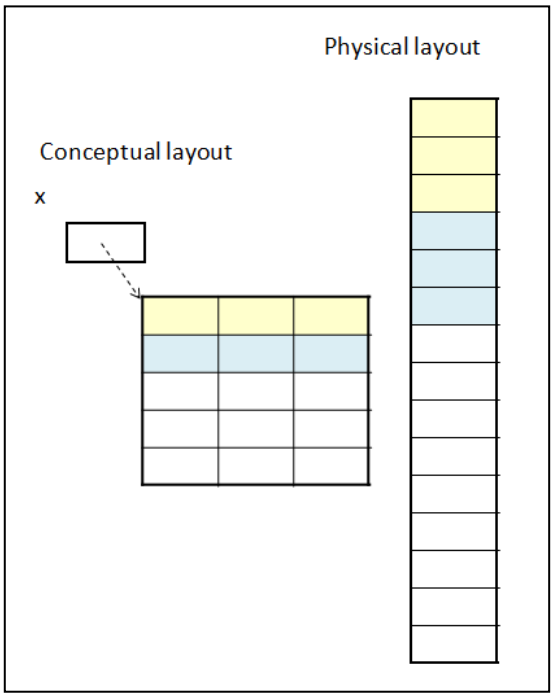

```
5 3
1 3 2
7 1 2
6 5 3
2 3 7
6 0 4
```

```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 void release(int _____________, int m)
05 {
06      for (int i=0; i<m; i++)
07          _____________
08      _____________
09 }
10
11 int main()
12 {
13      int i, j, m, n;
14      int _____________;
15
16      scanf("%d%d", &m, &n);
```

```
17
18      x = _____ malloc(_____*m);
19      for (i=0; i<m; i++)
20          x[i] = _____ malloc(_____*n);
21
22      for (i=0; i<m; i++)
23          for (j=0; j<n; j++)
24              scanf("%d", _____);
25
26      // some processing on data stored in array x
27
28      release(x, m);
29      return 0;
30 }
```

Sol:

```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 void release(int **x, int m)
05 {
06      for (int i=0; i<m; i++)
07          free(x[i]);
08      free(x);
09 }
10
11 int main()
12 {
13      int i, j, m, n;
14      int **x;
15
16      scanf("%d%d", &m, &n);
17
18      x = (int **) malloc(sizeof(int *)*m);
19      for (i=0; i<m; i++)
20          x[i] = (int *) malloc(sizeof(int)*n);
21
22      for (i=0; i<m; i++)
23          for (j=0; j<n; j++)
24              scanf("%d",&x[i][j]);
25
26      // some processing on data stored in array x
27
28      release(x, m);
29      return 0;
30 }
```



(b) [10] 請參考題 (a) 的功能描述，完成下列程式片段以右圖的方式動態配置第二種架構的二維陣列 x，用來存放上題串流裡讀入的 m 列 n 行整數矩陣，同樣請完成 release(x) 函式來釋放這種方法配置的陣列。

```
01 #include <stdio.h>
02 #include <stdlib.h>
03
```

```
04 void release(int _________)
05 {
06         _____________;
07 }
08
09 int main()
10 {
11         int i, j, m, n;
12         int _____________;
13
14         scanf("%d%d", &m, &n);
15
16         x = (___________) malloc(_____________*m);
17
18         for (i=0; i<m; i++)
19               for (j=0; j<n; j++)
20                     scanf("%d", _________________);
21
22         // some processing on data stored in array x
23
24         release(x);
25         return 0;
26 }
```

Sol:

```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 void release(int ____(*x)[3]____) // or x[5][3] or x[][3] or (*const x)[3]
05 {
06         ____free(x);____
07 }
08
09 int main()
10 {
11         int i, j, m, n;
12         int ____(*x)[3]____;
13
14         scanf("%d%d", &m, &n);
15
16         x = ____(int (*)[3])____ malloc(____sizeof(int [3])____*m);
17
18         for (i=0; i<m; i++)
19               for (j=0; j<n; j++)
20                     scanf("%d",____&x[i][j]____);
21
22         // some processing on data stored in array x
23
24         release(x);
25         return 0;
26 }
```

4. (a) [12] 在設計遞迴函式時我們談過分治法 (Divide and Conquer)，在實習的時候也練習寫過不需額外記憶體(in place)的快速排序法 (Quick Sort)，有的時候在應用中我們不需要把整個數列依照順序排好，只需要找到其中第 k 大的元素，此時可以用和快速排序法幾乎一樣的分

治法來遞迴完成，請完成下列 quickSelect() 遞迴函式，由一個沒有特定順序的數列中找到第 k 大的元素：(下面程式假設我們已經有一個函式 int partition(int d[], int start, int end) 以指定的元素為中心將數列在同一個陣列中分為兩半，指定元素會放在由小到大順序排列的正確位置上，在指定元素前面的都小於它，在指定元素後面的都大於等於它，partition() 函式回傳指定元素被放在陣列的哪一個索引上)

```
01 int quickSelect(int k, int d[], int start, int end) //資料在陣列 d[start], d[start+1], …, d[end]
02 {
03      int t, pivot;
04      if (end-start > 1)
05      {
06          pivot = _____________________
07          if (pivot>k-1)
08              return _____________________
09          else if (pivot<k-1)
10              return _____________________
11      }
12      else if (end-start == 1)
13      {
14          if (d[start]>d[end])
15              t=d[end], d[end]=d[start], d[start]=t;
16      }
17      return k-1;
18 }
19
20 int main()
21 {
22      int k=8, ans, data[]={-8, 20, -25, -15, 18, -49, 29, 15, 13, 9}, ndata=10;
23      ans = quickSelect(k, data, 0, ndata-1);
24      printf("the %d-th element is %d\n", k, data[ans]);
25      return 0;
26 }
```

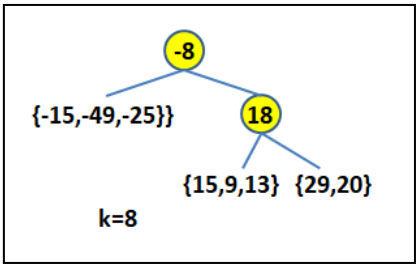
Sol:

```
06      pivot = ____partition(d, start, end);____
07      if (pivot>k-1)
08          return ____quickSelect(k, d, start, pivot-1);____
09      else if (pivot<k-1)
10          return ____quickSelect(k, d, pivot+1, end);____
```

(b) [4] 上面這個 quickSelect() 函式執行時列印出 **the 8-th element is 18**，資料陣列中許多元素會被修改，請問至少有哪些元素會在正確順序的位置上? _____ (原因為何)

Sol:

-8 and 18 Quick selection is a partition-based divide and conquer algorithm, if the k-th element is in the first half of the sequence, the algorithm only dig into the first half of the sequence. Therefore, as shown in the figure on the right, if we draw the process of partition into a tree, only the path directly leading to the k-th element will be put into their correct places.

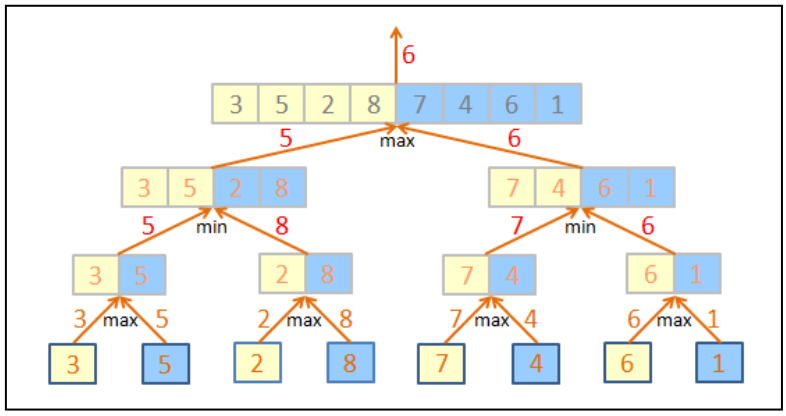


(c) [4] 請問第 12 至第 16 列似乎目的只在排出正確的順序，這樣子對於找到第 k 個元素有什麼幫助?

Sol:

If there are only two elements left in the range with k being one of them, since the k-th element would be in its correct place at the end of the algorithm, the other one must also be in its correct place.

5. 在一個遊戲中有 $n=2^k$ 個角色，每個角色都有自己的分數，大家可以藉由這個分數找出優勝者，但是這個過程並不是單純地比較大小，n 個角色首先分為左半邊 n/2 個和右半邊 n/2 個，分別找到 n/2 個的優勝者，這兩個優勝者可以決定何者勝出；對於左(右)半邊 n/2 個又繼續平分為左半邊 n/4 個以及右半邊 n/4 個，也是分別找到 n/4 個的優勝者，兩半邊的優勝者可以決定整體 n/2 個的優勝者；這個找尋優勝者的過程一直持續下去到不能再平分、只剩下 1 個角色時停止，他就是該組的優勝者。決定優勝的方法在最上層是取左右兩個裡面較大者，第二層是左右兩個裡面較小者，第三層是左右兩個裡面較大者，如此交替一直到最底層。以上圖 $n=2^3$ 個角色為例：先分為左右兩群 {3,5,2,8}, {7,4,6,1}、個別再往下細分下去一直到每一群只有 1 個角色，此時 3 和 5 比大、5 勝出，2 和 8 比大、8 勝出，7 和 4 比大、7 勝出，6 和 1 比大、6 勝出，往上一層則是 5 和 8 比小、5 勝出，7 和 6 比小、6 勝出，最上層是 5 和 6 比大、6 勝出，最後優勝者的分數是 6。由於 n 個角色的問題可以拆成左右兩個 n/2 個角色的問題，最後優勝者可以由子問題的優勝者進一步得到，因此非常適合設計遞迴函式來解，請撰寫一個遞迴函式計算這個優勝的分數：



(a) [2] 第 06 列請定義遞迴函式 findScore() 來計算一組角色比對後的最後優勝分數，它的參數包括此組的角色個數 n、存放分數的整數陣列 x[] 的起始位址、以及一個整數參數 type 代表最上層決定優勝的方法 (1 代表最大值、0 代表最小值)，函式回傳值為獲得優勝的分數

(b) [2] 第 09 列請檢查遞迴的結束條件

(c) [2] 第 10,11 列請遞迴呼叫 findScore() 並且傳入適當的參數計算左右兩側的優勝分數

(d) [2] 第 13,15 列請根據 type 參數計算並回傳本組的優勝分數

(e) [2] 第 25 列請完成 main() 函式內遞迴函式 findScore() 的呼叫

```
01 #include <stdio.h>
02
03 int max(int x, int y) { return x>y ? x : y; }
04 int min(int x, int y) { return x<y ? x : y; }
05
06 __________ findScore(__________, __________, __________)
07 {
08      int left, right;
09      if (_____________) return _____________;
10      left = findScore(_____, _____, _____);
11      right = findScore(_____, _____, _____);
12      if (type)
13          return _____________
14      else
```

```
15              return _________________
16 }
17
18 int main()
19 {
20      int i, n, x[1024];
21      while (1==scanf("%d", &n))
22      {
23          for (i=0; i<n; i++)
24              scanf("%d", x+i);
25          printf("%d\n", findScore(_______, ________, _______));
26      }
27      return 0;
28 }
```

Sol:

```
01 #include <stdio.h>
02
03 int max(int x, int y) { return x>y ? x : y; }
04 int min(int x, int y) { return x<y ? x : y; }
05
06 int findScore(int n, int x[], int type)
07 {
08      int left, right;
09      if (n==1) return x[0];
10      left = findScore(n/2, x, 1-type);
11      right = findScore(n/2, x+n/2, 1-type);
12      if (type)
13          return max(left,right);
14      else
15          return min(left,right);
16 }
17
18 int main()
19 {
20      int i, n, x[1024];
21      while (1==scanf("%d", &n))
22      {
23          for (i=0; i<n; i++)
24              scanf("%d", x+i);
25          printf("%d\n", findScore(n, x, 1));
26      }
27      return 0;
28 }
```

(f) [5] 上面這個遞迴函式寫得很精簡，結果也是正確的，但是實際測試起來卻覺得速度有點慢，仔細分析一下會發現在 $n=2^k$ 的時候遞迴函式呼叫的深度是 k+1，如果能夠使得呼叫的深度變成 k，遞迴函式呼叫的總次數幾乎少掉一半，因為函式裡面沒有什麼花時間的運算，少一半的呼叫會使得執行的速度幾乎縮短一半，請問該如何調整遞迴結束的條件來達成?

Sol:

```
05 int findScore(int n, int x[], int type)
```

```
06 {
07        int left, right;
08        if (n==1) return x[0];
09        if (n==2)
10              left = x[0], right = x[1];
11        else
12              left = findScore(n/2,x,1-type),
13              right = findScore(n/2,x+n/2,1-type)
14        if (type)
15              return max(left,right);
16        else
17              return min(left,right);
18 }
```

(g) [3] 請定義一個 2 個元素的函式指標陣列 fn[2] 並且初始化為 min() 及 max() 函式

Sol:

int (*fn[2])(int, int) = {min, max};

(h) [3] 請運用 (g) 的函式指標陣列來取代第 12 到第 15 列如下：

return _____(left, right);

Sol:

return **fn[type]**(left, right); or return **(*fn[type])**(left, right);