1091 NTOUCSE 程式設計 1C 期中考參考答案

109/11/10 (二)

考試時間：**13:20 – 16:00**

考試規則：1. **請閱上課本**，除了印給你的之外**不可**參考任何文件包括小考、作業、實習、或是其它參考資料
2. 你可以在題目卷上直接回答，可以使用鉛筆，但是**請在答案卷上註明題號**
3. 你覺得有需要的話，可以使用沒有教過的語法，但是**僅限於 C/C++ 語言**
4. 程式撰寫時請寫完整的程式碼，寫 。。。 的分數很低 (一個程式裡有重複很多遍的敘述是扣分的)
5. **不可**使用電腦、平板、智慧手機、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. (a) [7]請問下面程式執行的輸出為何?

   (b) [3]請問哪些變數的定義不符合 ANSI C89 語法(這是目前最普遍被 C 編譯器支援的標準)?

   (c) [5]哪些變數是區域變數? 哪些是全域變數? 哪些是靜態變數?

   (d) [5]請說明第 4 列及第 13 列所定義兩個變數的可見範圍 (scope) 以及生命期 (lifetime)

```
01 #include <stdio.h>
02 int a = 100;
03 int fun(int r) {
04         static int a = r;
05         return a = a+r;
06 }
07 int main() {
08         int b = 100;
09         {
10                 int a = 10;
11                 printf("%d ", a);
12         }
13         int c = fun(a);
14         printf("%d %d ", a, fun(a+5));
15         for (int a=0; a<2; a++) {
16                 char c = 'a';
17                 printf("%d %c ", a, c++);
18         }
19         return 0;
20 }
```

Sol:

(a) 10 100 305 0 a 1 a

(b) C89 標準是目前最普遍被支援的標準，它允許在每一個區塊 (左右大括號圍起來的區域) 的開始、可以執行的敘述之前定義區域變數，所以第 13 列的 int c 和第 15 列的 int a 都是不允許的，但是 C99, C11, 或是 C++98/03/11/14/17/20 都是允許而且提倡使用的。

(c) 區域變數：第 3, 8,10,13,15,16 列
全域變數：第 2 列
靜態變數：第 4 列

(d) 第 4 列 static int a=r;
scope 是第 4,5 列
lifetime 是在第 13 列第一次呼叫 fun 以後就存在，一直到程式執行到 19 列結束
第 13 列 int c = fun(a);

scope 是第 13,14,**15**,19 列

lifetime 是在第 13 列定義 c 以後就存在，一直到程式執行到 19 列結束

2. (a) [6] 請以小括號及強制型態轉換標示下面第 2 列運算式 ch*s-d/f-(s+i/2)的順序運算?
('a' 的 ASCII 內碼是 97)

(b) [4] 請問列印的結果為何?

```
01 char ch='a'; short s=2; int i=3; float f=2.0f; double d=6.28;
02 printf("%.2f\n", ch*s-d/f-(s+i/2));
```

Sol:

(a) ((double) ((int)97*(int)2) – (6.28 / (double) 2.0)) – (double) ((int)2 + (3/2))

(b) (double) 194 – 3.14 – (double) (2+1) = 187.86

3. [5] 請完成下面判斷一個 long long 整數 n 是否為 3 的指數次方的函式，是回傳 1 否回傳 0

```
01  int isPowerOf3(long long n) {
02        if (n==0) return 0;
03        while ( _______%3==0 ) ________
04        return ____________;
05  }
```

Sol:

while (n %3==0) n /= 3 ;

return n==1 ;

4. (a) [5] 請寫出下列三種定義/初始化字元陣列的方法所定義的字元陣列的長度以及字元陣列初始的內容： ① char str[] = {'j', 'o', 'e'}; ② char str[4] = {'j', 'o', 'e'}; ③char str[] = "joe";

(b) [5] 請問這三種方法下面第 3 列的輸出為何?

(c) [5] 哪一種方法會有機會讓下面第 4 列印出亂碼? 請解釋印出亂碼的原因?

```
01. int i;
02. for (i=0; i<4; i++)
03.       printf("%d ", str[i]); //  字元常數 'j', 'o', 'e' 的 ASCII 內碼分別為 106 111 101
04. printf("%s", str);
```

Sol:

(a)

① 編譯器處理 char str[] = {'j', 'o', 'e'}; 敘述的時候，因為 str[]沒有指定陣列有幾個元素，編譯器會自己計算大括號裡面元素的個數，所以會定義一個有 3 個元素的字元陣列，定義的時候內容依序為字元 'j', 'o', 'e' 的內碼 106, 111, 101

② 編譯器處理 char str[4] = {'j', 'o', 'e'}; 敘述的時候，因為 str[4]已經指定字元陣列有 4 個元素，所以這是一個 4 個元素的字元陣列，因為大括號裡面少了一個元素，編譯器預設會補 0，定義的時候內容依序為字元 'j', 'o', 'e' 的內碼 106, 111, 101, 以及編譯器補上的 0, char str[4] = {'j', 'o', 'e', '\0'};

③ 編譯器處理 char str[] = "joe"; 敘述的時候，因為 str[]沒有指定陣列有幾個元素，編譯器會自己計算 "joe" 常數字串裡面有幾個字元，但是編譯器會看到常數字串最後有一個額外的結束字元 '\0'，所以會定義一個有 4 個元素的字元陣列，定義的時候內容依序為字元 'j', 'o', 'e', '\0' 的內碼 106, 111, 101, 0

(b) 第 3 列在三種定義方法下輸出會是

① 106 111 101 ??? ②106 111 101 0 ③ 106 111 101 0

上面 ??? 代表內容不確定，因為陣列 str 只定義了 3 個元素，所以不能確定列印第 4 個元素的時候會看到什麼數值，甚至有可能執行到的時候因為想要存取不該用的記憶體而當掉。

(c) 第 4 列運用 printf("%s" 來列印，printf 函式在處理 %s 格式化的命令時，對於後面所傳入的字元陣列預設是列印到內容為 '\0' 的字元才會停下來，也就是它裡面會有一個 while (data[i++]!='\0') {.....} 的迴圈，由於在第①種定義方法裡面，我們只能確定前三個自原有資料，第四個字元以及後續的內容無法確定，所以 printf 自己往後列印一直到記憶體裡面剛好出現 0 的地方為止，所以通常會看到亂碼，有的時候會因為想要存取不該用的記憶體而當掉。

雖然你一開始會覺得這三種寫法差異很小，沒有仔細分析的時候你可能會覺得 C 程式有點難預期，覺得第 1 種方法會有莫名其妙的錯誤，也分不太出來差異在哪裡，但是仔細分析過以後，你應該要注意到其實這個差異是可以預期的，當初在看到 int x[] = {1,2,3}; 的敘述時，有說編譯器會自己去算有幾個元素，在看到 int x[10] = {1,2,3}; 的敘述時，也有說大括號裡面不夠的 7 個元素編譯器會補 0，在看到 char str[] = "hello"; 的時候有說這是編譯器為了讓你少打一些單引號像是 char str[] = {'h', 'e', 'l', 'l', 'o', '\0'}; 所以特別幫你處理字串常數 "hello"，並不能用在其它地方，如果你這三個地方都有注意到，就會得到上面的認知，同時也會覺得 C 程式的表現是完全可以預期的。另外也請注意上面這些都是定義陣列並且同時初始化陣列的時候才可以使用的，不要很快樂地覺得自己可以舉一反三了，把 x[] = {1,2,3};、x[10] = {1,2,3}; 或是 str[] = "hello"; 獨立出來使用，程式的語法是一個有限的集合，沒有講的沒有看到的就是沒有，不要自己生出新的語法。

5. 考慮右側的輸入資料，每一列是一筆測資，測資有兩種格式，第一種是一個整數 x，第二種是一個字元 c 再接著一個整數 x，不管是那一種資料，程式需要處理整數 x，例如下面程式第 3 列的列印 x，請回答下列問題：

```
範例輸入
n12
3456
7
p89
10
```

```
01  int x; char c;
02  while (c='␣', 1==scanf("%d", &x) || 2==scanf("␣%c%d", &c, &x)) {
03          printf("Type %d: data is %d\n", 1+(c!='␣'), x);
04  }
```

上面程式中 ␣ 代表一個空格

(a) [5] 請解釋上面 while 迴圈處理輸入的運作原理？(右側為 C 的運算子優先順序表格)

Precedence	Operator	Associativity
1	++ -- () [] . -> (type){list	Left-to-right
2	++ -- + - ! ~ (type) * & sizeof Alignof	Right-to-left
3	* / %	Left-to-right
4	+ -	
5	<< >>	
6	< <= > >=	
7	== !=	
8	&	
9	^	
10	|	
11	&&	
12	||	
13[1]	?:	Right-to-Left
14	= += -= *= /= %= <<= >>= &= ^= |=	
15	,	Left-to-right

Sol:

這個寫法用到了幾個概念：

1、首先一個運算式(expression)裡可以有逗點這個運算子，也就是右圖運算子優先順序最低 15 的那個，逗點運算子的結合律是由左到右，定義了連續兩個逗點執行的順序，但是一個逗點兩邊的算式執行的順序其實也是有定義的，逗點是一個 sequence point，一個 sequence point 之前所有的 side effects 一律需要更新到變數裡，然後才繼續執行，也就是逗點左邊的會先做完，這裡所說的逗點是在一般的運算式裡面的逗點，不包括同時定義多個變數時的逗點 int x,y,z;、陣列變數定義時初始化的逗點 int x[3]={1,2,3};、還有函式呼叫的參數的逗點 fun(x, y, z)，所以 c='␣' 會比兩個 scanf 先執行。每一個運算式在計算完的時候都會有一個數值，以逗點相隔的運算式的數值是最後一個逗點後面的數值，例如運算式 a = (x, y, z); 時 a 的數值就是 z 的數值。為什麼要在 while() 的判斷運算式這裡加上 c='␣', 的運算式呢？目的是希望在每次執行接下來的兩個 scanf 之前能夠先把字元變數 c 的內容設為空格，如此就算接下來只執行到 1==scanf("%d", &x)，變數 c 裡面一定會是空格。另一種寫法是

```
c='␣'; // 執行第一次 scanf 之前設定字元變數 c 為空格
while (1==scanf("%d", &x) || 2==scanf("␣%c%d", &c, &x)) {
    …
    c='␣'; // 在執行下一次 scanf 之前重設字元變數 c 為空格
}
```

這個寫法有一模一樣的功能，只是 c='␣'; 要寫兩次而已。

2、上面這個寫法還用到第二個概念是邏輯 || 運算子「短路(short circuit)執行」的特性，由於邏輯的「或」運算只要兩側有一個為真，計算的結果就為真，所以執行的時候如果左邊的運算 1==scanf("%d", &x)已經是真，右邊的運算 2==scanf(" %c%d", &c, &x) 就不會執行了，因此上面的敘述會先嘗試讀取十進位數字進入變數 x，如果輸入串流裡的確是一個十進位數字，scanf("%d" 會回傳成功完成格式轉換的次數 1，此時並不會執行 || 右側的 scanf("␣%c%d"；反之如果輸入串流裡是一個字母 (例如 n12 的 n)，因為並不是 0~9 的數字，這個時候 scanf("%d" 會失敗，輸入串流還是停在這個字母，scanf(…) 回傳成功完成格式轉換的次數 0，這個時候 1==scanf(…)和 1 比較會失敗，就會去執行 || 右側的 scanf("␣%c%d", &c, &x)，將字母讀入字元變數 c 中，將緊接在後面的數字讀入整數變數 x 中，並且回傳成功完成格式轉換的次數 2。注意在 && 邏輯運算式以及 || 邏輯運算式裡 && 和 || 也是 sequence point。在串流結束時，|| 左側的 scanf 會先執行並且回傳 EOF (-1)，1==scanf(…)和 1 比較會失敗，於是執行 || 右側的 scanf，此時還是回傳 EOF (-1)，2==scanf(…)和 2 比較也會失敗，整個運算值是 0。

(b) [5] 請寫出上面程式片段執行時的輸出?

Sol:

```
Type 2: data is 12
Type 1: data is 3456
Type 1: data is 7
Type 2: data is 89
Type 1: data is 10
```

6. 下面為一個計算 x^n 的遞迴函式

```
01 double power(double x, int n) {
02     if (n==0) return 1.;
03     return x * power(x, n-1);
04 }
```

(a) [4] 請問一個函式具有什麼性質可以稱為遞迴函式?

Sol:

一個函式裡面如果呼叫自己來完成要求，或是呼叫其它函式，其它函式又呼叫自己這個函式來完成要求，就稱為遞迴函式

(b) [4] 請問執行 double y = power(2, 14);，得到最後 y 的數值之前，總共會呼叫 power(double, int) 函式幾次?

Sol:

依序呼叫 power(2,14), power(2,13), power(2,12), power(2,11),…, power(2,0) 共 15 次

(c) [4] 請完成下面的 power2()遞迴函式，此函式與 power()函式功能完全一樣，請問執行 double y = power2(2, 14); 總共會呼叫 power2(double, int) 函式幾次?

```
01 double power2(double x, int n) {
02     if (n==0) return 1;
03     if (n%2==1)
```

```
04              return x * power2(x, _____);
05          else
06              return power2(x*x, _____);
07 }
```

Sol:

return x * power2(x, n-1);
return power2(x*x, n/2);
依序呼叫 power2(2,14), power2(4,7), power2(4,6), power2(16,3), power2(16,2), power2(256,1), power2(256,0) 共 7 次

(d) [4] 請完成下列 power3()函式，維持功能與 power(), power2() 相同，繼續降低遞迴呼叫的次數，請問執行 double y = power3(2, 14); 總共會呼叫 power3(double, int) 函式幾次?

```
01 double power3(double x, int n) {
02          if (n==0) return 1;
03          double x2 = power3(____, ____);
04          x2 *= x2;
05          if (n%2==1)
06              return _______;
07          else
08              return x2;
09 }
```

Sol:

double x2 = power3(x , n/2);
return x * x2 ;
依序呼叫 power3(2,14), power3(2,7), power3(2,3), power3(2,1), power3(2,0) 共 5 次

(e) [4] 請完成下列 power4()函式，藉由「條件運算子?:」較為簡潔地完成遞迴函式的撰寫

```
01 double power4(double x, int n) {
02          if (n==0) return 1;
03          return (_____?_____:_____) * power4(_______, _______);
04 }
```

Sol:

return (n%2==1 ? x : 1) * power4(x*x , n/2);

7. [10] 請撰寫一個程式讀取右圖輸入串流中的測資，每一筆測資一列，包含 x n_1 n_2 三個資料，例如 147 10 2，其中字串 x 代表一個 n_1 進位制的正整數，這個整數的數值會小於 10^{18}，請用 n_2 進位制輸出 x，其中 $2 \le n_1, n_2 \le 16$。

範例輸入	範例輸出
147 10 2 11011 2 10 12345 6 15 AA5 13 8	10010011 27 845 3441

Sol:

```
#include <stdio.h>
int main() {
    int i, base1, base2;
    long long num;
    char num1[65], num2[65];
    while (3==scanf("%s%d%d", &num1, &base1, &base2))
    {
        for (num=0,i=0; num1[i]!=0; i++)
            if ((num1[i]>='0')&&(num1[i]<='9'))
```

```
                num = num*base1 + num1[i]-'0';
            else if ((num1[i]>='A')&&(num1[i]<='F'))
                num = num*base1 + num1[i]-'A'+10;
        for (i=0; num>0; num/=base2,i++)
        {
            num2[i] = num%base2;
            if (num2[i]>10)
                num2[i] += 'A'-10;
            else
                num2[i] += '0';
        }
        for (i--; i>=0; i--)
            printf("%c", num2[i]);
        printf("\n");
    }
    return 0;
}
```

8. [10] 請撰寫一個程式，讀入兩個字串 s 與 t，如下範例輸入：s 在第一列，t 在第二列，其中 t 是將 s 裡面所有字元順序隨機打亂，同時任意挑選一個位置加入任意一個小寫英文字元所得到，s 與 t 字串中字元都是小寫英文字母，s 的長度可以為 0 到 100000000 的任意整數，請輸出 t 中新增的字元。

範例輸入	範例輸出
ajlkhkfmaxnmvdfhauierhqwrew khearhjuiwmearwkxdfhlfmvanqa	a

Sol:

```
#include <stdio.h>
int main()
{
    int c, i, s[26]={0}, t[26]={0};
    while (((c=getchar())>='a')&&(c<='z'))
        s[c-'a']++;
    while (c!='\n') c=getchar(); // 假設 s 與 t 之間只有 '\r' '\n' (etutor) 或是 '\n'
    while (((c=getchar())>='a')&&(c<='z'))
        t[c-'a']++;
    for (i=0; i<26&&t[i]==s[i]; i++);
    printf("%c\n", 'a'+i);
    return 0;
}
```

while (c!='\n') c=getchar(); 也可以用下面的 scanf 敘述取代，在 etutor 目的是讀一個換列字元，在其他機器上則完全不讀任何一個字元

```
scanf("%*1[\n]");
```

這個題目不適合使用字元陣列配合 scanf("%s"、gets()或是 fgets()來讀取資料，因為題目要求的 s 字串的長度太長了，而且所需要完成的功能其實不需要把整個字串記錄下來，這個題目的精神在於資料表示方法的轉變，只需要使用兩個 26 個元素的整數陣列來表示這兩個陣列所使用到的字元就足夠來找出新加入的字元了。

也可以只用一個 26 個元素的整數陣列就夠了，程式如下：

```
#include <stdio.h>
int main()
```

```
{
	int c, i, s[26]={0};
	while (((c=getchar())>='a')&&(c<='z'))
		s[c-'a']++;
	scanf("%*1[\n]");
	while (((c=getchar())>='a')&&(c<='z'))
		s[c-'a']--;
	for (i=0; i<26&&0==s[i]; i++);
	printf("%c\n", 'a'+i);
	return 0;
}
```

還有另外一種想法也可以很簡單地完成題目的要求，就是把所有 s 字串裡面的字元加總起來，把 t 字串裡面的字元加總起來，後者減去前者就是新加入字元的 ASCII 內碼，這樣子寫的時候需要留意加總的數字的範圍，10^8*('z'-'a')=10^8*25=2,500,000,000，這個數字剛好比 2^{31}-1=2,147,483,647 大了一些，可以用 unsigned long 也可以用 long long 型態的變數，程式如下：

```
#include <stdio.h>
int main()
{
	int c;
	long long sum=0;
	while (((c=getchar())>='a')&&(c<='z'))
		sum -= c-'a';
	while (c!='\n') c=getchar();
	while (((c=getchar())>='a')&&(c<='z'))
		sum += c-'a';
	printf("%c\n", 'a'+sum);
	return 0;
}
```

這個方法在現在這個題目裡比用陣列紀錄出現次數還簡單，但是如果題目變成加入一個英文單字 (例如 hello) 的話就比較難還原了。