C Language

1. point

- 解釋：指標也算是一種變數，只是裡面存的不是一般的數值，而是記憶體位置

```c
#include <stdio.h>

void func(int *ptr){
    (*ptr)++;
}

int main(){
    int num = 5;
    func(&num);
    printf("num:%d\n",num);
    return 0;
}
```

- Q1：解釋以下指標意義

```c
int a;
int *a;
int **a;
int a[10];
int *a[10];
int [*a](10);
int (*a)(int);
int (*a[10])(int);

// 一個整數型別
// 一個指向整數的指標
// 一個指向指標的指標，而"指向的指標"是指向一個整數型別
// 一個有10個整數型的陣列
// 一個有10個指標的陣列，該指標是指向一個整數型別
// 一個指向有10個整數型陣列的指標
// 一個指向函數的指標，該函數有一個整數型參數並返回一個整數
// 一個有10個指標的陣列，該指標指向一個函數，該函數有一個整數型參數並返回一個整數
```

- Q2：寫一個function讓變數a跟b能夠交換，不透過暫存變數

```c
#include <stdio.h>

void swap(int *a, int *b) {
    *a = *a ^ *b;
    *b = *a ^ *b;
    *a = *a ^ *b;
}

int main() {
    int a = 5;
    int b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
```

- Q3：回答printf的答案

```c
#include <stdio.h>

int main(void) {
    char *str[] = {
        {"MediaTekOnlineTesting"},
        {"WelcomeToHere"},
        {"Hello"},
        {"EverydayGenius"},
        {"HopeEverythingGood"}
    };
    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    char* m = str[4] + 4;
    char* n = str[1];
    char* p = *(str+2) + 1;
    int *q = &(a + 1)[3];
    printf("1. %s\n", *(str+1));
    printf("2. %s\n", (str[3]+8));
    printf("3. %c\n", *m);
    printf("4. %c\n", *(n+3));
    printf("5. %c\n", *p + 1);
    printf("6. %d\n", *q);
    return 0;
}
// 1. WelcomeToHere
// 2. Genius
// 3. E
// 4. c
// 5. f
// 6. 5
```

- Q4：回答陣列的答案

```c
#include <stdio.h>

int main() {
    int a[] = {6, 7, 8, 9, 10};
    int *p = a;
    *(p++) += 100;
    *(++p) += 50;
    for (int i = 0; i < 5; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }
}
// a[0] = 106
// a[1] = 7
// a[2] = 58
// a[3] = 9
// a[4] = 10
```

2. call by value, call by reference, call by address

- *call by value：參數以數值方式傳遞，複製一份到另一個呼叫此參數的副程式*

> *(C 語言只有 call by value)*

```c
#include <stdio.h>

int swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

int main(void)
{
    int a = 5;
    int b = 10;
    swap(a, b);
    printf(" a = %d b = %d", a, b); // a = 5,b = 10
}
```

- Call by reference：將參數以數值的方式傳遞到呼叫此參數的副程式，副程式需要有一個參考來接收這個參數。
  註：傳參考是C++才有的東西，C語言是沒有的唷!

```
void swap(int &c, int &d)
{
    int temp = c;
    c = d;
    d = temp;
}
int main()
{
    int a = 5, b = 10;
    swap(a, b);
    printf(" %d %d ", a, b);
    return 0;
}

// 在 C 語言中沒有Call by reference,故編譯時會有錯誤
```

- Call by address : 參數以記憶體位置的方式傳到呼叫此參數的副程式，副程式需要有一個指標來指到這個參數的記憶體位置
  *註 : Call by addres本質上也是call by value，只不過那個value剛好就是address而已

```
#include <stdio.h>

int swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main(void)
{
    int a = 5;
    int b = 10;
    swap(&a, &b);
    printf(" a = %d b = %d", a, b);// a = 10, b = 5
}

// 透過指標去傳遞位址(Call by address)，可將函數內的變數作為指向該變數的指標傳遞給函數，從而讓函數內的變
```

3. variable scope and lifetime

- local 變數 : local 變數僅活在該函式內，存放位置在 stack 或 heap 記憶體中。

```c
#include <stdio.h>

void count()
{
    int c = 1;
    printf("%d\n", c); // c = 1 c = 1...
    c++;
}

int main(void)
{
    for(int i = 0; i < 10; i++)
        count();
    return 0;
}
```

- static 變數 : static 變數生命周期 (life time) 跟程式一樣長，而範圍 (scope) 則維持不變，即在宣告的函式之外仍無法存取 static 變數。

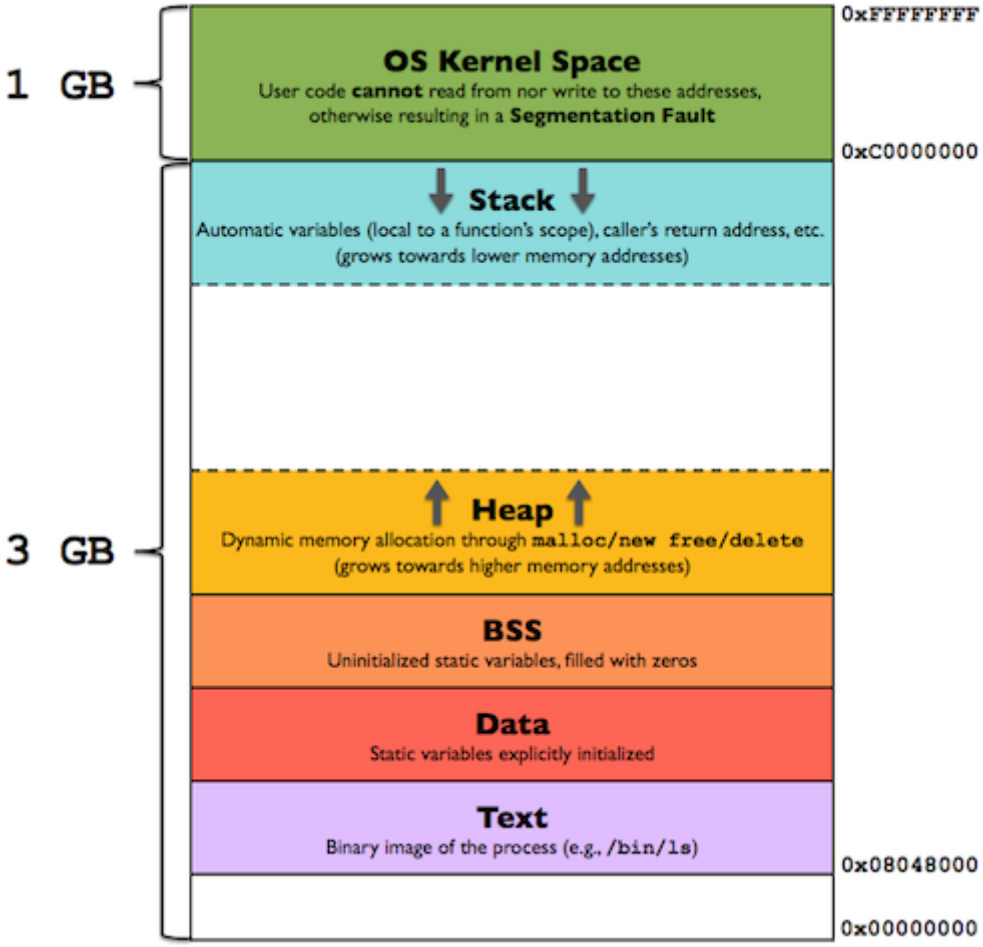
```c
#include <stdio.h>

void count()
{
    static int c = 1;
    printf("%d\n", c); // c = 1 c = 2...
    c++;
}

int main(void)
{
    for(int i = 0; i < 10; i++)
        count();
    return 0;
}
```

- global 變數 : 所有區段皆可使用此變數。

```c
#include <stdio.h>

int c = 1;

void count()
{
    printf("%d\n", c); // c = 1 c = 2...
    c++;
}

int main(void)
{
    for(int i = 0; i < 10; i++) {
        count();
    }

    return 0;
}
```

- 記憶體的配置

Stack : 存放函數的參數、區域變數等，由空間配置系統自行產生與回收，會稱作 stack 是由於其配置遵守 LIFO (Last-In-First-Out)。
Heap : 一般由程式設計師分配釋放，執行時才會知道配置大小，如 malloc/new 和 free/delete。
Global : 包含 BSS (未初始化的靜態變數)、data section (全域變數、靜態變數) 和 text/code (常數字元)。



4. extern

extern 用於聲明外部變數，如果變數需要在多個文件中共享，就需要使用 extern 聲明該變數；如果變數只在當前文件中使用，可以不使用 extern 關鍵字聲明。

```
// var.c
int a = 10;


// main.c
#include "var.c"

extern int a;

int main() {
    printf("a = %d\n", a); //a = 10
    return 0;
}
```

5. const

const 通常表示只可讀取不可寫入的變數，常用來宣告常數。使用const有以下好處：

1. 使編譯器保護那些不希望被改變的參數。
2. 編譯器處理方式 : define 在預處理階段展開；const 在編譯階段使用。
3. 類型和安全檢查 : const 會在編譯階段會執行類型檢查，define 則不會。
4. 存儲方式 : define 直接展開不會分配記憶體，const 則會在記憶體中分配。

```
#include <stdio.h>

int main(void)
{
    int n1 = 10, n2 = 100;

    const int* ptr1 = &n1; // ptr1是一個指向const的整數指標
    *ptr1 = 20; // X
    ptr1 = &n2; // O (ptr1 = 100)

    int* const ptr2 = &n1; // ptr2是一個指向整數的const指標
    *ptr2 = 20; // O (ptr2 = 20)
    ptr2 = &n2; // X
}
```

6. volatile

編譯器不對 volatile 修飾的變數進行最佳化處理,而是每次都去讀取變數實際上最新的數值。

volatile 常見的應用：

1. 修飾中斷處理程式中(ISR)中可能被修改的全域變數。
2. 修飾多執行緒(multi-threaded)的全域變數。
3. 設備的硬體暫存器(如狀態暫存器)

```
extern const volatile unsigned int rt_clock;
```

這是在 RTOS kernel 常見的一種宣告：rt_clock通常是指系統時鐘，它經常被時鐘中斷進行更新。所以它是volatile。

7. inline

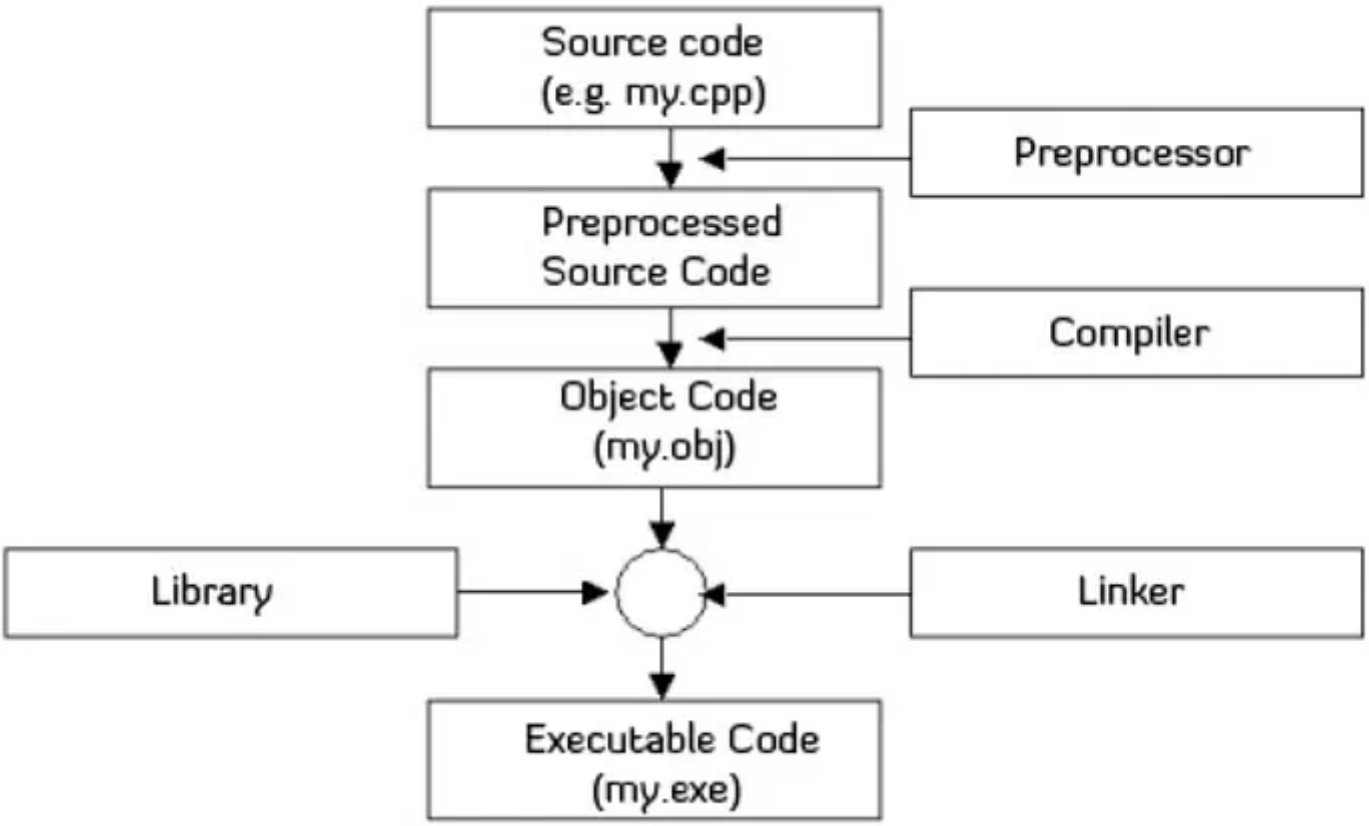
- inline function寫法 :

```
inline int square(int x)
{
  return x * x;
}
```

- Macro寫法 :

```
# define SQUARE(x) ((x) * (x))

printf("\n %d", SQUARE(5)); // 若在主程式中，下述能得到 25，看起來沒有問題

printf("\n %d", SQUARE(3+2)); // 但如果是以下，卻會得到 11 (3+2 * 3+2)
```

- 差異 :
  macro : 在前置處理器(preprocessor)階段時,直接進行文字替換
  inline : 在編譯(compile)階段時,直接取代function



8. #define

#define 是巨集，在前置處理器(preprocessor)執行時處理，將要替換的程式碼展開做文字替換。define 語法範例如下：

```
#define PI 3.1415926 //常數巨集
#define A(x) x //函數巨集
#define MIN(A,B)  ((A) <= (B) ? (A) : (B))
```

- 引入防護和條件編譯：

引入防護 (Include guard) 是一種條件編譯，用於防範 #include 指令重複引入的問題。

```
#ifndef MYHEADER // 避免重複引入
#define MYHEADER
...
#endif
```

9. different between pointer and array

就記憶體方向來看，指標所用的記憶體位置不為連續，而"陣列"所配置的空間為連續。

10. different between interrupt and polling

Interrupt 和 Polling 差異在中斷具有即時性,而輪循是定時檢查

interrupt : 當中斷觸發時,處理器會暫停目前處理的任務,轉而去執行中斷相關程序,等到處理完畢時才會回到原本的任務上。

polling : 它是一種定時檢查的方式,當檢查到有事件發生時才會去執行它。

17. bubbleSort

- bubbleSort範例：

```c
#include <stdio.h>

int bubbleSort(int arr[],int n){
    for(int i=0; i<n-1 ;i++){
        for(int j=i+1; j<n; j++){
            if(arr[i] > arr[j]){
                int temp = arr[j];
                arr[j] = arr[i];
                arr[i] = temp;
            }
        }
    }
}

int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    for (int i=0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

18. struct

struct 是使用者自定的型態，包含數個不同資料型態的變數，將不同的資料型態關聯在一起，使他們的關聯更直覺。

- struct佔幾byte

```
typedef struct MyStruct
{
    char a[2];
    int  b;
    double c;
};
//ans = 1+3(對齊int)+4+8=16byte

typedef struct MyStruct
{
    char a[2];
    int  b;
    double c;
    int *Pint;
    char d;
    char*Pchar;
};
//ans = 1+3(對齊int)+4+8+4+1+3+4(由於(4+1+3+4)不是8的倍數故需要補4)+4=32byte

typedef struct MyStruct
{
    char a;
    char b;
    struct str2
    {
        int d;
    } c;
};
//ans = 1+1+4+2(對齊int)=8byte

typedef struct MyStruct
{
    char a;
    int b;
    char c;
    char* d;
    double* e;
    struct str2
    {
        int f;
        char g;
        struct str3
        {
            char* p;
        }n;
    } m;
};
//ans = 1+3+4+1+3+4+4+4+1+3+4=32byte

typedef struct MyStruct
```

```
{
    char a;
    char b;
    struct str2
    {
        char c;
        char d;
    };
};
//ans = 1+1=2byte  *並沒有申明這個結構體的變數所以str2不用計算
```

19. union

在語法結構上，union與 struct 類似，都是使用者自定義的資料結構。但union 與 struct 差別最大的地方，就在於 union 結構中的各變數是共用記憶體位置。
並且在任何時候，只有一個變數的值是有效的，這取決於最後一次賦值的變數。

struct 稱為結構體，可以包含數個不同資料型態的變數，所有變數佔用不同的記憶體。
union 稱為聯合體，也可以包涵不同資料型態的變數，但是所有變數佔用相同記憶體。

- union範例：

```
union myUnion {
    int i;
    float f;
    char c;
};

int main() {
    union myUnion u;
    u.i = 42;
    printf("u.i = %d\n", u.i);  // output : u.i = 42
    u.f = 3.14;
    printf("u.f = %f\n", u.f);  // output : u.f = 3.140000
    u.c = 'A';
    printf("u.c = %c\n", u.c);  // output : u.c = A
    printf("u.i = %d\n", u.i);  // output : u.i = 1092616192
    //在對 f 和 c 進行賦值之後，u.i 的值也發生了變化，這是因為 i、f 和 c 共享同一塊記憶體
    return 0;
}
```

- union應用：

```c
//1. 存儲不同類型的數據

#include <stdio.h>
#include <stddef.h>

struct {
    char c;
    int i;
    double d;
} myStruct;

union {
    char c;
    int i;
    double d;
} myUnion;

int main() {
    printf("Size of struct: %lu\n", sizeof(myStruct));   // output : 16
    printf("Size of struct members: %lu %lu %lu\n", sizeof(myStruct.c), sizeof(myStruct.i), size

    printf("Size of union: %lu\n", sizeof(myUnion));   // output : 8
    printf("Size of union members: %lu %lu %lu\n", sizeof(myUnion.c), sizeof(myUnion.i), sizeof(

    return 0;
}

//2. 位元應用
union flags {
    struct {
        unsigned int flag0 : 1;
        unsigned int flag1 : 1;
        unsigned int flag2 : 1;
        unsigned int flag3 : 1;
        unsigned int : 4; /* 4 bits reserved */
    } bits;
    unsigned char value;
};

int main() {
    union flags f;
    f.value = 255;
    f.bits.flag0 = 0;
    f.bits.flag1 = 0;
    printf("f.bits.flag0 = %d\n", f.bits.flag0);   // output : 0
    printf("f.bits.flag1 = %d\n", f.bits.flag1);   // output : 0
    printf("f.bits.flag2 = %d\n", f.bits.flag2);   // output : 1
    printf("f.bits.flag3 = %d\n", f.bits.flag3);   // output : 1
```

```
    return 0;
}
```

20. enum

enum 是一種常數定義方式，可以提升可讀性，enum 裡的識別字會以 int 的型態，從指定的值開始遞增排列 (預設為 0)

- enum範例：

```
enum color1 {red, green, blue, yellow};
// red = 0, green = 1...

enum color2 {red=10, green, blue=20, yellow};
//red = 10, green = 11, blue = 20, yellow = 21
```

41. sizeof

- 各型態大小

```
64-bit / 32-bit
sizeof(string)       : 8   : 4
sizeof(char)         : 1   : 1
sizeof(p)            : 8   : 4
sizeof(short)        : 2   : 2
sizeof(int)          : 4   : 4
sizeof(long)         : 8   : 4
sizeof(long long)    : 8   : 8
sizeof(size_t)       : 8   : 4
sizeof(double)       : 8   : 8
sizeof(long double)  : 16  : 12
```

- 二維陣列大小

```c
#include <stdio.h>
int main()
{
    int arr[3][4];
    printf("1:%d\n", sizeof(arr));  // 結果為 48 ( 3 * 4 * 4(int) )
    char str[2][10];
    printf("2:%d\n", sizeof(str));  // 結果為 20 ( 2 * 10 * 1(char) )
    int* ptr[5][3];
    printf("3:%d\n", sizeof(ptr));  // 結果為 120 ( 5 * 3 * 8(int*) )
    struct Point
    {
        int x;
        int y;
    };
    struct Point points[4][3];
    printf("4:%d\n", sizeof(points));  // 結果為 96 ( 4 * 3 * 8(struct Point) )
    return 0;
}
```

6. bit operation

setting a bit

```c
int set_bit(int x, int n)
    return x | (1 << n);
```

clearing a bit

```c
int clear_bit(int x, int n)
    return x & ~(1 << n);
```

fliping a bit

```c
int flip_bit(int x, int n)
    return x ^ (1 << n);
```

checking a bit

```c
int check_bit(int x, int n)
    return (x >> n) & 1;
```

回答ans的數值

```c
#include <stdio.h>

int main() {
    long ans = 0;
    short a = 0x1234;
    short b = 0x5600;
    ans += a << 16;
    ans += b << 0;
    ans += (b >> 2) + 0x22;
    printf("a=%x\n",ans);   // 12346BA2
}
```

1. 設定一個絕對位址為0x67a9的整數型變數的值為0xaa55

```c
#include <stdio.h>

int main(void)
{
    int *ptr = (int *)0x67a9;
    printf("p1=%x\n", ptr); // p1=67a9
    ptr = (int *)0xaa55;
    printf("p2=%x\n", ptr); // p2=aa55
}
```

2. N是否為判斷2的次方

```c
int isPowerof2(int n) {
    return n > 0 && (n & (n - 1)) == 0;
}
```

4. 連續呼叫 func 10 次，印出的值為何？

```c
#include <stdio.h>

void func(void)
{
    static int i = 0 ;  //若沒static，i會一直歸零
    i++ ;
    printf("%d" , i ) ;
}

int main(void)
{
    for(int i=0; i<10; i++)
        func(); // 12345678910
}
```

5. i++ & ++i

```c
#include <stdio.h>

int main()
{
    int i = 5;
    int j = i++ // i++ : 表示先返回 i 的值，然後再將其加 1。
    printf("i1 = %d\n",i);  // 6
    printf("j1 = %d\n",j);  // 5

    i = 5;
    j = ++i; // ++i : 表示先將 i 的值加 1，然後再返回它。
    printf("i2 = %d\n",i);  // 6
    printf("j2 = %d\n",j);  // 6
    return 0;
}
```

10. 寫個function判斷基數偶數

```c
if (num % 2 == 0) {
    printf("even\n");
} else {
    printf("odd\n");
}
```

11. 寫個function計算有幾個位元是 1

```
int func(int x){
    int sum=0;
    while(x){
        x &= x-1;
        sum++;
    }
    return sum;
}
```

13. define vs typedef

```
#define dPS struct s *
typedef struct s * tPS;
```

以上兩種情況的意圖都是要定義dPS 和 tPS 作為一個指向結構s指標。哪種方法更好呢 ? (如果有的話)為什麼 ?

答案是：typedef 更好。思考下面的例子：

```
dPS p1, p2;
tPS p3, p4;
```

第一個擴展為

```
struct s * p1, p2;
```

上面的程式碼定義p1為一個指向結構的指標，p2為一個實際的結構，這也許不是你想要的，所以沒事不要亂用 macro。

14. What is the output of the following program

```
int main() {
    unsigned int a = 6;
    int b = -20;
    (a + b > 6) ? puts(">6") : puts("<= 6");
}

// 當表達式中存在有符號類型和無符號類型（unsigned）
// 時所有的操作數都自動轉換為無符號類型。
// 因此-20變成了一個非常大的正整數，所以該表達式計算出的結果大于6。
```

15. The faster way to an integer multiply by 7

```
int main() {
    int i = 1;
    i = (i << 2) + (i << 1) + (i << 0);
    printf("i = %d\n",i);
}
```

16. declaration（宣告）和 definition（定義）的差異

- declaration（宣告）
  宣告是指告訴編譯器一個變數、函數、類型等識別符號的名稱和類型，
  但不分配存儲空間，也不執行初始化。例如，以下是一個變數的宣告：

```
extern int x;
void foo();
struct MyStruct;
```

- definition（定義）
  定義是指為變數、函數、類型等識別符號分配存儲空間，
  並且可能進行初始化。例如，以下是一個變數的定義：

```
int x;
void foo() {
    //content
}
struct MyStruct {
    //content
};
```

在C語言中，每個變數和函數只能有一個定義，但可以有多個宣告。

17. Reverse a string

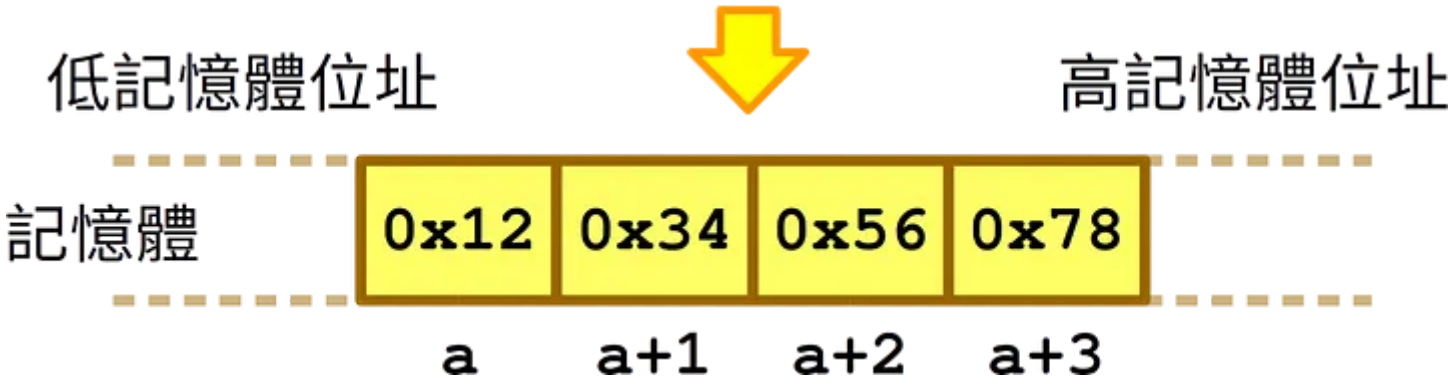
```c
#include <stdio.h>
#include <string.h>

void reverseString(char *str) {
    int left = 0;
    int right = strlen(str) - 1;
    while (left < right) {
        char temp = str[left];
        str[left] = str[right];
        str[right] = temp;
        left++;
        right--;
    }
}

int main() {
    char str[] = "ABCDE";
    reverseString(str);
    printf("反轉後字串: %s\n", str);
    return 0;
}
```

20. 判斷Big-Endian or Little-Endian

在現今主流的 CPU 中，最常見的位元組順序有兩種，分別是 Big-Endian 與 Little-Endian，Big-Endian 是指資料放進記憶體中的時候，最高位的位元組會放在最低的記憶體位址上，而 Little-Endian 則是剛好相反，它會把最高位的位元組放在最高的記憶體位址上。

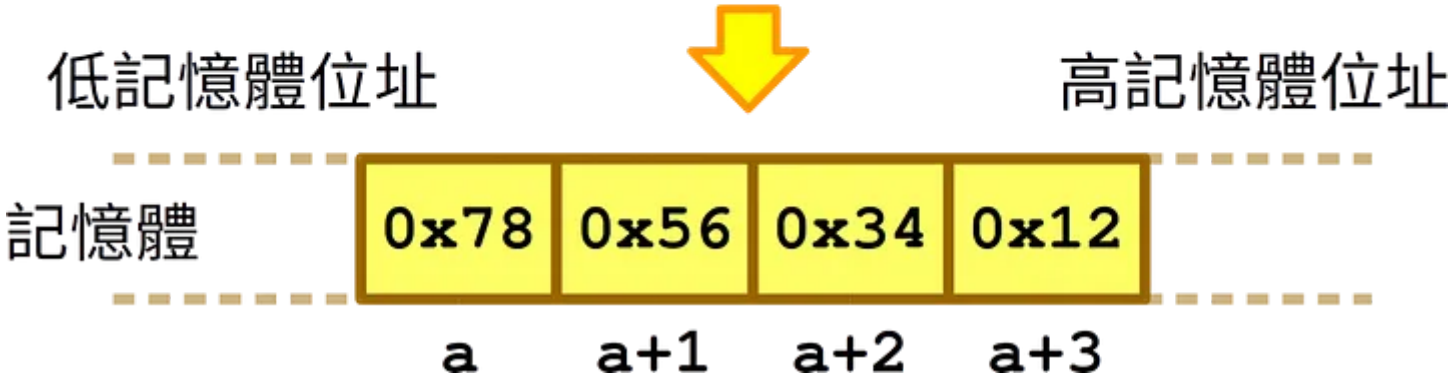
Big-Endian

0x12345678



Little-Endian

0x12345678



21. 給一個int a[20]已排序的陣列，請寫一個function(a, size)能印出0~500的數字，且不包含a陣列內的元素

```c
#include <stdio.h>

void function(int *a,int size)
{
 for(int i=0; i<=500; i++)
 {
    if(i == *a)
        a++;
    else
        printf("%d\n",i);
    }
}
```

22. 分類0~500

```c
void function(int *a, int size, int b)
{
    int *ptr = a;
    int i;

    while (*ptr < b * 100) {
        ptr++;
    }

    for (i = b*100; i<(b+1)*100; i++) {
        if (*ptr == i)
             ptr++;
        else
            printf("%d\n", i);
    }
}
```

23. 印出下列圖形

```
    *
   **
  ***
 ****
*****
```

```
#include <stdio.h>

int main()
{
    for(int i=0; i<5; i++){
        for(int j=0+i; j<5; j++)    printf(" ");
        for(int j=5-i; j<=5; j++)   printf("*");
        printf("\n");
    }
}
```

24. a是多少

```
unsigned int a,b;

for(b = 2; b >= 0 ; b--)
{
   a++;
}
cout << a;
```

Ans : 這題因為b是unsigned int 所以永遠不會小於0，你就回答爆掉或a = inf就好

25. 費式數列

0 1 1 2 3 5 8 13 21 34 55...n,寫一個函數,輸入值是位置的值"n",要找出相對應的值

```c
#include <stdio.h>

int fibonacci(int n) {
    if (n <= 1)
        return n;

    int prev = 0;
    int current = 1;
    int next;

    for (int i = 2; i <= n; i++) {
        next = prev + current;
        prev = current;
        current = next;
    }

    return current;
}

int main() {
    int n = 10;  // 要計算費式數列的項數
    printf("費式數列的前 %d 項：\n", n);
    for (int i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }
    printf("\n");

    return 0;
}
```

26. quick sort

```
void quick_sort(int arr[], int first_index, int last_index) {
  // 宣告索引變數
  int pivotIndex, temp, index_a, index_b;

  if (first_index < last_index) {
    // 以第一個元素作為基準
    pivotIndex = first_index;
    index_a = first_index;
    index_b = last_index;

    // 以遞增方式排序
    while (index_a < index_b) {
      while (arr[index_a] <= arr[pivotIndex] && index_a < last_index) {
        index_a++;
      }
      while (arr[index_b] > arr[pivotIndex]) {
        index_b--;
      }

      if (index_a < index_b) {
        // 交換元素
        temp = arr[index_a];
        arr[index_a] = arr[index_b];
        arr[index_b] = temp;
      }
    }

    // 交換基準元素與 index_b 元素
    temp = arr[pivotIndex];
    arr[pivotIndex] = arr[index_b];
    arr[index_b] = temp;

    // 遞迴呼叫快速排序法函數
    quick_sort(arr, first_index, index_b - 1);
    quick_sort(arr, index_b + 1, last_index);
  }
}
```

27. binary serach

```c
#include <stdio.h>

int binary_search(int arr[], int left, int right, int target){
    while(left <= right){
        int mid = left + (right - left);
        if(arr[mid] == target)
            return mid;
        else if(arr[mid] > target)
            right = mid - 1;
        else
            left = mid + 1;
    }
    return -1;
}

int main(){
    int arr[] ={1,2,3,5,7,9,12,18,22};
    int n = sizeof(arr)/sizeof(arr[0]);
    int result = binary_search(arr,0,n-1,2);
    printf("res=%d\n",result);
    return 0;
}
```

28. 給一個unsigned short, 問換算成16進制後,四個值是否相同? 若是回傳1,否則回傳0

```c
int isHexEqaul(unsigned short input) {     // input = 0xAAAA;
    unsigned short hex[4]; // 0xFFFF ==> 0000 0000 0000 0000
    int is_eqaul;

    hex[0] = (input & 0xF000) >> 12;
    hex[1] = (input & 0x0F00) >> 8;
    hex[2] = (input & 0x00F0) >> 4;
    hex[3] = input & 0x000F;

    is_eqaul = (hex[0] ^ hex[1] ^ hex[2] ^ hex[3]);
    return !(is_eqaul);
}
```

29. 求一個數的最高位1在第幾位

```c
#include <stdio.h>

int max_bit(int bit){
    int cnt = 0;
    while(bit){
        cnt++;
        bit >>= 1;
    }
    return cnt - 1;
}

int main(){
    printf("%d",max_bit(2));
    return 0;
}
```

30. 最大公因數 遞迴寫法

```c
#include <stdio.h>

int gcd(int a, int b) {
    if(b == 0)
        return a;
    return gcd(b,a%b);
}

int main() {
    printf("GCD: %d\n", gcd(15,5));

    return 0;
}
```

31. (多選題)問哪邊會出現錯誤(以及錯在哪)，及要如何初始化?

```c
union Var
{
    char ch;
    int num1;
    double num2;
};
int main()
{

    union Var var = {123};
    printf("var.ch = %c\n",var.ch);
    printf("var.num1 = %d\n",var.num1);
    printf("var.num2 = %.3f\n",var.num2);

    return 0;
}

// 初始化時給予的數值類型不符合 union 成員的類型。
// %c是印出字元,%f是印出浮點數,而var是整數所以他們兩個錯。
```

33. 0~500個數字每次隨機 取一個數字出來，但下次在抽出時不可以出現已經抽過的數字，問你如何時實現。

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_NUM 500

int random(int *arr) {
    int count = 0; // 已經抽出的數字個數
    while (count < MAX_NUM) {
    int num = rand() % (MAX_NUM + 1); // 隨機產生一個數字
        if (arr[num] == 0) { // 如果這個數字還沒被抽過
            arr[num] = 1; // 標記為已經抽過
            count++; // 已經抽出的數字個數加 1
            printf("%d ", num); // 輸出這個數字
        }
    }
}

int main() {
    int nums[MAX_NUM + 1]; // 儲存 0~500 的數字，初始化為 0
    for (int i = 0; i <= MAX_NUM; i++) {
        nums[i] = 0;
    }
    random(nums);
    return 0;
}
```

34. #define MIN(A,B) A < B ? A:B; 這樣會有甚麼問題?

```c
int result = 2 * MIN(6,10); // 2*6<10?6:10;
//result 會變成10，但我們要的是12
```

36. 解釋C語言中的 "#error"

在C語言中，預處理指令（preprocessor directives）以 '#' 字符開。

#error指令會讓預處理器產生一個錯誤消息，並停止編譯。它通常被用來在預處理時檢測錯誤或者不符合要求的條件，比

```
//這是一個防止編譯器編譯這個文件時，如果沒有定義DEBUG這個macro，預處理器會產生一個錯誤消息
#ifndef DEBUG
#error "You must define the DEBUG macro"
#endif
```

37. Explain lvalue and rvalue

lvalue：指的是可以出現在等號左邊 (assignment expression 的左邊) 的運算式，lvalue 可以被當成是一個物件

rvalue：指的是不能出現在等號左邊的運算式，也就是沒有記憶體位置 (address) 的運算式，只有值 (value) 的運算

以下是一例子： `int a = 5;  // a 是 lvalue，5 是 rvalue`

38. 寫一個“標準”巨集MIN ，這個巨集輸入兩個參數並返回較小的一個。

```
#include <stdio.h>

#define MIN(a,b) (a < b ? a : b)

int main()
{
    printf("min=%d\n",MIN(10,5));
}
```