Q1: 考慮以下 C99 程式，解釋其具體作用，並用 for/while 迴圈改寫，隨後提供 uint16\_t 的版本。在什麼場合會用到下方的程式碼？

#include <stdint.h>

uint32\_tfunc(uint32\_t x) {

uint32\_t n = x;

n = ((n & 0xffff0000) >> 16) | ((n & 0x0000ffff) << 16);

n = ((n & 0xff00ff00) >> 8) | ((n & 0x00ff00ff) << 8);

n = ((n & 0xf0f0f0f0) >> 4) | ((n & 0x0f0f0f0f) << 4);

n = ((n & 0xcccccccc) >> 2) | ((n & 0x33333333) << 2);

n = ((n & 0xaaaaaaaa) >> 1) | ((n & 0x55555555) << 1);

return n;

}

顛倒位元順序

一開始是前16bits跟後16bits交換

接著是每個16bits中的前8 bits跟後8bits做交換  
以此類推

#include <stdint.h>

uint32\_tfunc(uint32\_t x) {

uint32\_t n = x;

inti,j,k;

int shift, mask;

for ( i=0; i<=4; i++ )

{

shift = 1 <<i;

mask = 1;

for ( j=0; j<32; j=j+(shift\*2) )

{

for ( k=0; k<shift; k++ )

{

mask = mask<<1;

mask++;

}

mask = mask<<shift;

}

n = ((n & mask) >> shift) | ((n & ~mask) << shift);

}

return n;

}

uint16\_t 的版本

#include <stdint.h>

uint16\_tfunc(uint16\_t x) {

uint16\_t n = x;

inti,j,k;

int shift, mask;

for ( i=0; i<=3; i++ )

{

shift = 1 <<i;

mask = 1;

for ( j=0; j<16; j=j+(shift\*2) )

{

for ( k=0; k<shift; k++ )

{

mask = mask<<1;

mask++;

}

mask = mask<<shift;

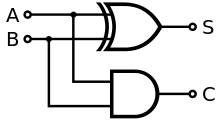
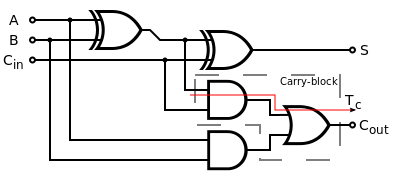
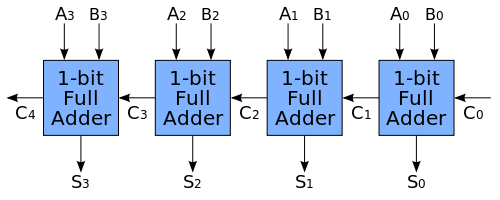
}

n = ((n & mask) >> shift) | ((n & ~mask) << shift);

}

return n;

}

* Q2: 在 C 程式中，使用遞迴和 bit-wise operator 來實作乘法運算，請參考以下提示:
* 加法器是用於執行加法的電路元件，通常由 AND 閘、OR 閘和 XOR 閘構成
  + 也可用加法器實作減法，只要將減數轉成二補數，並注意溢位即可
* 半加器：將兩個一位二進位數相加 (input: A, B) (output: S, C)  
  
* 全加器：將兩個一位二進位數相加 (input: A, B, Cin) (output: S, Cout)  
  
* 波紋進位加法器：使用多個一位全加器構成 N 位加法器  
  
* 半加器可用以下 C 程式來實作:

uint32\_thalf\_add(uint32\_t a, uint32\_t b) {

if (b == 0) return a;

uint32\_t sum = a ^ b; /\* 相加但不進位 \*/

uint32\_t carry = (a & b) << 1; /\* 進位但不相加 \*/

returnhalf\_add(sum, carry);

}

uint32\_tmul(uint32\_t a, uint32\_t b) {

if (b == 1) return a;

if (b == 0) return 0;

if (b & 1)

returnhalf\_add(a ,(mul( a, b>>1 )<<1));

else

returnhalf\_add(0 ,(mul( a, b>>1 )<<1));

}

* Q3:思考以下 C 程式的用途，以及在什麼場合用得到 (提示: 記憶體管理常式)，探討應用場合時，需要一併列出最小可編譯和運作的 C 程式碼。

char \*p;

...

\*p = (\*p) & ~1;

想不出來用途……..只看出來他拿指到的現有值去掉最低的bit

* Q4: 考慮以下 C 程式在 GNU/Linux 中，透過 linked list 來實作動態記憶體管理 (malloc和 free)，虛擬記憶體的使用如下圖，初步的程式如下方，要注意到程式碼並不完整，也不能在多執行緒環境安全運用。請改寫 malloc 程式碼使其正確運作，並提供對應的 free 實作。

目前僅完成單執行序態記憶體管理

Q5: 假設下方 C 程式檔名為 fork.c，在 GNU/Linux 上編譯得到名為 fork 的執行檔，我們可用 ./fork | wc -c 計算輸出的 - 字元，請解釋程式行為和輸出的 - 字元數量的關聯。

在fork中，子程序會從呼叫fork地方開始執行，將所有可能畫出來可以知道僅考慮fork出的個數一共是

2+4+8=14個，但實際上，執行會發現- 字元數量比我們預期的還多，主要的原因是來自printf，printf一般不會馬上把結果往stdout送，而是會等buffer滿了才送，在這個情形fork產生子程序時，也會將父程序的printf buffer複製一份，故會有多印的情形。

