\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Report: HW4

Author: E64061151 林友鈞 <[E64061151@mail.ncku.edu.tw](mailto:wang@xiaoming.tw)>

Class: 資訊系二乙

Description:

這份作業讓我學到如合適當的使用函數，以及如何在函數間傳遞參數。在重寫的過程也發現hw2的bug，原來在for裡忘記break，導致多計算了很多次，這讓我明白函數的重要性，因為使用函數在程式中比較容易除錯。

第一部分將解釋hw4\_2.c所有函數的功能以及使用方式。

int SetData(int data[],const char \*choice, int P)

這個函數的主要目的是去設定ans和guess。

int data[] :

是一個整數型態的指標，可以ans或是guess。

const char \*choice :

是一個字元陣列(字元型態的指標)需要輸出給使用者現在是需輸入”ans: ”或是”guess: ”

int P :

由command line讀入的P。

首先這個函數會輸出ans: 或是guess: 看使用者傳入函數的choice是什麼。接下來這個函數會讀取P個數字。最後清空buffer，避免影響下一次輸入。這個函數會這樣設計，是因爲ans和guess都需要輸入，如果不寫成函數，那整體程式的開發會增加許多時間，而且也較不好維護。

int Check(int data[],int H,int P)

這個函數的主要目的是去檢查ans是否有符合規定。

int data[] : 是一個整數型態的指標，傳入的會是ans。

int H : 由command line讀入的H。

int P : 由command line讀入的P。

這個函數會從在data中scan P個元素，只要有一個元素小於1或是大於H就會回傳0

當所有元素都掃過，就會回傳1，代表data所有的元素都符合規則。

int \* Zeroing(int data[],int size)

這個函數做的是陣列的歸零雖然可以使用 int a[10] = {0};的方式歸零，但是這樣每次需要歸0過的陣列都需要重新宣告。

int data[] : 傳入的是需要歸零的陣列。

int size : 傳入的是 data陣列的大小。

int ComputeH(int \*ans\_match, int \*ans, int \*guess\_match int \* guess, int P)

這個函數做的事情是去計算H的值。

int \*ans\_match :

ans\_match是一個一維陣列，他要記錄的值是ans中第幾個元素在計算H時有和guess match。

int \*ans :

ans 是使用者輸入的解答，是一個一維陣列。

int \* guess\_match :

guess\_match是一個一維陣列，他要記錄的值是guess中第幾個元素在計算H時有和ans match。

int \* guess :

guess 是使用者輸入的答案，會去與ans做比對。

int P : 由command line讀入的P。

這個函數會Scan ans 和 guess，只要元素在陣列中的位置相同且值也一樣，那h++。假設ans的第i個元素和guess的第i個元素一樣，那ans\_match[i] = guess\_match[i] = 1。這樣做是避免在ComputeX函數中重複計算。

int ComputeX(int \*ans\_match, int \*ans, int \*guess\_match int \* guess, int P)

這個函數做的事情是去計算X的值。

int \*ans\_match :

ans\_match是一個一維陣列，他記錄的值是ans中第幾個元素在計算H時有和guess match。

int \*ans :

ans 是使用者輸入的解答，是一個一維陣列。

int \* guess\_match :

guess\_match是一個一維陣列，他記錄的值是guess中第幾個元素在計算H時有和ans match。

int \* guess :

guess 是使用者輸入的答案，會去與ans做比對。

int P : 由command line讀入的P。

這個函數的功能是去計算X。程式邏輯是這樣的，有一個兩層的for loop會去逐一比對，從guess[0]開始到guess[P]，每一個在計算H時沒有配對成功的guess的元素都會和所有還沒有配對成功的ans元素做比對。一旦比對到的值相同則x++，ans\_match一樣會像計算H時紀錄下來，並break避免guess的元素再和後面ans的元素配對(作業二就是忘記break QQ)避免重複計算。迴圈結束後將X回傳。

void Reset(int \*ans\_match, int \*guess\_match, int \* guess, int P)

這個函數主要做的是當整個遊戲將要進入新的循環(使用者輸入 guess，計算H和X時)進行的重新設定。

int \* ans\_match :

記錄每次ans配對成功的陣列，在遊戲前需要歸零，避免上個循環影響下個循環。

int \* guess\_match :

記錄每次guess配對成功的陣列，在遊戲前需要歸零，避免上個循環影響下個循環。

int \*guess :

需要讓使用者從新輸入。

int P :

由command line讀入的P。

ans\_match 和 guess\_match再函數中會被傳入Zeroing函數中做歸0，guess則會傳入SetData函數中讓使用者從新輸入。

void PlayGame(int \*ans, int P)

這個函數做的事情是去維持這個遊戲的進行。

int \* ans :

在使用者設定的ans。

int P :

由command line讀入的P。

這個函數的流程是這樣Reset -> ComputeH -> ComputeX -> 輸出結果，如果H不等於P代表答案不完全正確，則再重新再一次這個循環，直到H==P也就是使用者輸入的guess==ans為止。

int \* SetAns(int \*ans, int P, int N)

這個函數會去做讓使用者輸入ans並檢查。

int \*ans : 使用者欲填入ans的陣列。

int P : 由command line讀入的P。

int N : 由command line讀入的N。

這個函數會不斷用SetData讓使用者輸入ans並用Check檢查，直到輸入正確為止。

函數最後回傳ans。

int main(int argc, char const \* argv[])

main函數是程式的進入點。

int argc : 為command line的參數個數

char const \* argc[] : 為command line 的參數，裡頭包含程式名稱。

main 函數裡做的事將遊戲需要的N和P由command line中讀入以及宣告變數，並將字串轉成數字。呼叫SetAns來設定ans之後呼叫PlayGame函數來進行遊戲。最後return 0結束程式。

==========================第一部分結束=========================

第二部分將解釋hw4\_3.c所有函數的功能以及使用方式。

void OutputBitPattern(long long \*x,int bits) :

long long \*x : 需要輸出成bit pattern的數

int bits : 需要告訴函數要輸出的bits數是多少

這個函數會輸出bit pattern，至於為什麼用long long pointer來存，是因爲這樣32bits表示，64bits也能表示，函式只需寫一次就好。函式裡的迴圈裡的j會從最左邊的1開始逐一往右移，ex:1000...0000下一次0100...0000。此時利用&運算就能知道\*x的bit pattern是如何了。

int inputBitpattern\_int\_x32(const char \*pattern,int bits) :

const char \* pattern : 輸入的是要轉成int的bit pattern。

int bits : 需要告訴函數輸入的是幾bits的 bit pattern

這個函式需要input bit pattern，還有bits數，會output 32bits的整數。邏輯是這樣的，sum的型態是int，從pattern字串的最左邊開始scan，只要是1就去計算他的次方數，並用sum去加總。

long long inputBitpattern\_int\_x64(const char \*pattern,int bits) :

const char \* pattern : 輸入的是要轉成long long 的bit pattern。

int bits : 需要告訴函數輸入的是幾bits的 bit pattern

這個函式需要input bit pattern，還有bits數，會output 64bits的整數，所以回傳值是long long型態。邏輯和intputBitpattern\_int\_x32一樣，只是sum的型態變成long long。

void Mode(int mode, const char \* input\_data)

這個函數會根據mode 的不同呼叫其他函數完成事情

int mode : 使用者欲輸入的mode。

const char \* input\_data :

從command line 輸入的第二個參數，可以試bit pattern 也能是數字，函數會跟據不同mode做不同的轉變。

這個函數雖然在main中就能實現，不一定要寫在main外，但是當程式不只做bit pattern的轉換，另外還做其他任務時，寫成函數會是比較好的作法。在函數中mode == 1, mode == 4 做的事是將bit pattern轉成int 和 float，事實上只呼叫inputBitpattern\_int\_x32和intputBitpattern\_int\_x64兩個函數，因為輸出的整數就會有和給定的bit pattern有一樣的bit pattern，這時候用一個float或是double的指標指向這個整數，最後printf再用\*取值運算子將這個pointer輸出，就會變成float 或是 double的數了。這個是我作業三和作業四的差別，不用IEEE 754算的方式。

int main(int argc, char \* argv[])

main函數是程式的進入點。

int argc : 為command line的參數個數

char const \* argc[] : 為command line 的參數，裡頭包含程式名稱。

在main中會先判斷argc是否< 3，如果小於則提出警示，並結束程式，否則呼叫Mode。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

hw4\_2 Code:

#include<stdio.h>

#include<stdlib.h>

int SetData(int data[],const char \*choice,int P){

int i;

printf("%s",choice);

for(i = 0; i < P; i++){

scanf("%d",&data[i]);

}

while ((getchar()) != '\n');// clear the buffer

return 1;

}

int Check(int data[],int H,int P){

int i = 0;

for(i = 0;i < P; i++){

if(data[i] < 1 || data[i] > H)

return 0; // if it is illegal the function return 0

}

return 1;// return 1 finally , means that all number is legal

}

int ComputeH(int \*ans\_match,int \*ans,int \*guess\_match,int \*guess,int P){

int h = 0;

for(int i = 0;i < P; i++){

if(guess[i] == ans[i]){

// if guess[i] match to ans[i] ans\_match[i] and guess\_match[i] are assigned 1

// this assignment is to avoid computing in "x" again

ans\_match[i]= guess\_match[i] = 1;

h++;

}

}

return h;

}

int ComputeX(int \*ans\_match,int \*ans,int \*guess\_match,int \*guess,int P){

int x = 0;

for(int i = 0; i < P; i++){

if(guess\_match[i] != 1){

for(int j = 0; j < P; j++){

if(ans\_match[j] != 1 && ans[j] == guess[i]){

ans\_match[j] = 1;

x++;

break;

}

}

}

}

return x;

}

int \* Zeroing(int data[],int size){

for(int i = 0;i < size; i++){

data[i] = 0;

}

return data;

}

void Reset(int \*ans\_match,int \* guess\_match,int \* guess,int P){

Zeroing(ans\_match,P); // reset ans\_match

Zeroing(guess\_match,P);// reset guess\_match

SetData(guess,"guess: ",P);//input guess

}

void PlayGame(int \*ans,int P){

int guess[P];

int ans\_match[P];

int guess\_match[P];

int h,x;

do{

h = 0;

x = 0;

Reset(ans\_match,guess\_match,guess,P);

h = ComputeH(ans\_match,ans,guess\_match,guess,P);

x = ComputeX(ans\_match,ans,guess\_match,guess,P);

printf("%dH %dX\n",h,x);

}while(h!=P);

}

int \* SetAns(int \*ans,int P,int N){

while(!SetData(ans,"ans: ",P)||!Check(ans,N,P))

printf("The answer is illegal, please reset the answer.\n");

return ans;

}

int main(int argc, char const \*argv[])

{

/\* code \*/

if(argc < 3){

printf("Arguments are not enough to do this program\n");

exit(0);

}else{

int N = atoi(argv[1]);

int P = atoi(argv[2]);

int ans[100];

printf("You set N=%d, P=%d\n",N,P);

SetAns(ans,P,N);

PlayGame(ans,P);

}

return 0;

}

Compilation:

gcc -std=c99 -o hw4\_2 hw4\_2.c

Execution:

./hw4\_2 5 5

Output:

You set N=5, P=5

ans:0 2 3 7 3

The answer is illegal, please reset the answer.

ans:2 3 4 5 5

guess:1 2 3 4 5

1H3X

guess:4 5 4 5 4

2H1X

guess:1 4 3 5 5

2H2X

guess:2 4 3 5 5

3H2X

guess:2 3 4 5 5

5H0X

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

hw4\_3 Code:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 學號：E64061151

\* 姓名：林友鈞

\* 編譯方式：gcc -o hw4\_3 hw4\_3.c

\* 執行方式：./hw4\_2 mode input

\* 程式功能：transfer bit pattern to number or transfer number to bit pattern

\* 更新日期：2018/10/28

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

// 以long long pointer 來存 位置 是因為這樣32bits 可以用 64bits也可以用

int OutputBitPattern(long long \*x,int bits){

printf("%d-bit: ",bits);

long long j = 0;

int i;

for(i = bits-1;i >= 0; i--){

j = 1;

printf("%d",\*x&j<<i?1:0);

}

printf("\n");

return 0;

}

// inputBitpattern\_int\_x32 和 inputBitpattern\_int\_x64 兩個function都是用來計算bit pattern 的整數值

// 兩者最大的不同是 sum 的型別

// 在x32 用int 這樣他就會自動overflow

// 在x64 用long long 也是同樣道理 當sign 是 1時就會自動overflow

int inputBitpattern\_int\_x32(const char \*pattern,int bits){

int sum = 0;

int i = 0;

int j = 1;

for(i = 0;i < strlen(pattern); i++){

if(pattern[i] == '1'){

// sum += ComputeExponent(2,(bits-1-i));// bit - 1 - i 是因為最右邊的0是0次方所以要減1，這行是在算2的幾次方

sum += j << (bits-1-i);

}

}

// printf("integer = %d\n",sum);

return(sum);

}

long long inputBitpattern\_int\_x64(const char \*pattern,int bits){

long long sum = 0;

int i = 0;

long long j = 1;

for(i = 0;i < strlen(pattern); i++){

if(pattern[i] == '1'){

// sum += ComputeExponent(2,(bits-1-i));

sum += j << (bits - 1 - i);

}

}

// printf("integer = %.0Lf\n",sum);

return(sum);

}

void Mode(int mode,const char \* input\_data){

switch(mode){

int x;

float y;

double z;

long long w;

long double r;

double \* test2;

float \* test;

case 1 :

x = inputBitpattern\_int\_x32(input\_data,32);

test = &x;

// printf("{\"integer\" : \"%d\",\"float\" : \"%f\"}\n",x,\*test);

printf("integer: %d\n",x);

printf("float: %f\n",\*test);

break;

case 2 :

x = atoi(input\_data);

OutputBitPattern(&x,32);

break;

case 3:

y = atof(input\_data);

OutputBitPattern(&y,32);

break;

case 4:

w = inputBitpattern\_int\_x64(input\_data,64);

test2 = &w;

// printf("{\"integer\" : \"%lld\",\"float\" : \"%lf\"}\n",w,\*test2);

printf("integer: %lld\n",w);

printf("float: %lf\n",\*test2);

break;

case 5 :

w = atoll(input\_data);

OutputBitPattern(&w,64);

break;

case 6 :

z = atof(input\_data);

OutputBitPattern(&z,64);

break;

}

}

int main(int argc, char \*argv[])

{

if(argc < 3){

double test = 1;

printf("count is not enough\n");

return 0;

}

else Mode(atoi(argv[1]),argv[2]);

return 0;

}

Compilation:

gcc -std=c99 -o hw4\_3 hw4\_3.c

Execution&output:

>> ./hw4\_3 1 00111111000011001100110011001101

>> integer: 1057803469

float: 0.550000

>> ./hw4\_3 2 1057803469

>> 32-bit: 00111111000011001100110011001101

>> ./hw4\_3 3 0.550000

>> 32-bit: 00111111000011001100110011001101

>> ./hw4\_3 4 1100001111011100010011111111011100101001111100110110010110100011

>> integer: -4333500818457926237

float: -8160483663085472768.000000

>> ./hw4\_3 5 -4333500818457926237

>> 64-bit: 1100001111011100010011111111011100101001111100110110010110100011

>> ./hw4\_3 6 -8160483663085472768.000000

>> 64-bit: 1100001111011100010011111111011100101001111100110110010110100011