Đáp án bài tập về nhà đầu tiên lớp C21:

*LƯU Ý: Đây là đáp án theo hướng suy nghĩ của mình, các bạn có hướng suy nghĩ khác mà đáp án đúng + đúng yêu cầu đều được chấp nhận *

Bài 1: Viết hàm in ra hình tam giác như sau trên màn hình:

```
    #
   ###
  #####
```

Bài này ta chỉ cần để ý một chút quy luật toán học là có thể làm được, đây là code của mình:

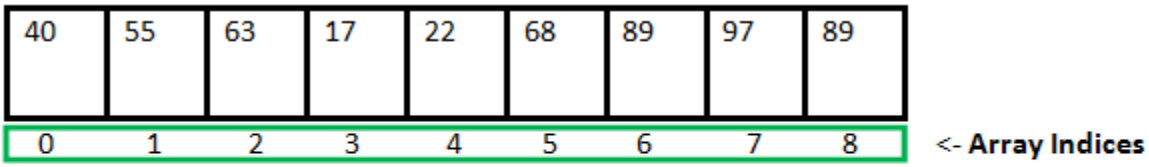
```c
#include <stdio.h>
void print_triangle(int n){
    for(int i = 0; i<n; i++){
        for(int j = 0; j<2*n -1; j++){
            if( ((n - i) < j)   &&  (j < (n + i)))  // This problem is quiet simple, all you need is a little math
            {
                printf("#");
            }

            else{
                printf(" ");
            }
    }
    printf("\n");
}
}
int main(){
    print_triangle(50);
    return 0;
}
```

Các bạn có thể thấy tầng dưới cùng có số dấu # là 2*n – 1, đồng thời số dấu # cũng giảm dần khi đi lên trên nên ta có câu if như hình, nếu không thỏa điều kiện thì ta in ra dấu cách " " để tạo hình tam giác.

Bài 2: Viết hàm thực hiện phép nhân vô hướng 2 vector, dạng vector đưa vào và trả về là mảng float có kích thước tùy ý.

Bài này khá đơn giản, điều mình muốn nói tới là về mảng và cách đưa mảng vào hàm.

Trong C thì khi mảng được tạo ra, một vùng bộ nhớ liền nhau sẽ được cấp phát, như hình sau:



Array Length = 9
First Index = 0
Last Index = 8

(trích https://www.geeksforgeeks.org/arrays-in-c-cpp/?ref=lbp)

Do đó thực chất khi truy xuất mảng, C sẽ tạo ra một biến Pointer (tức một biến chứa địa chỉ của một biến khác) để truy xuất dữ liệu từ mảng, hay nói cách khác:

MẢNG CÓ THỂ TRUY XUẤT BẰNG POINTER, TUY NHIÊN MẢNG KHÔNG PHẢI LÀ POINTER.

Do đó chúng ta có cách đưa mảng vào hàm như sau:

Nếu các bạn đã học qua C, thì đều sẽ biết đây gọi là "tham chiếu" hay reference (https://www.geeksforgeeks.org/references-in-c/) , tức chỉ địa chỉ bộ nhớ của mảng được đưa vào hàm, mọi thay đổi trên mảng này sẽ thay đổi trên địa chỉ được chỉ tới.

```c
#include <stdio.h>
float vector_mul(float *a, float*b, int size){
    float temp = 0;
    for(int i = 0; i<size; i++){
        temp += a[i]*b[i]; // This problem is not hard too, nothing too special
    }
    return temp;
}
int main(){
    float a[3] = {3.5,5,6};
    float b[3] = {1,2,3.4};

    float* v1 = a;
    float* v2 = a;
    float result = vector_mul(v1, v2, 3);
    printf("%f", result);
    return 0;
}
```

Tuy nhiên ta cũng có cách đơn giản hơn, nhưng thực ra là tương đương:

Kí hiệu float a[] ở trong trường hợp này là tương đương với float *a.

```c
#include <stdio.h>
float vector_mul(float a[], float b [], int size){
    float temp = 0;
    for(int i = 0; i<size; i++){
        temp += a[i]*b[i]; // This problem is not hard too, nothing too special
    }
    return temp;
}
int main(){
    float a[3] = {3.5,5,6};
    float b[3] = {1,2,3.4};

//  float* v1 = a;
//  float* v2 = a;
    float result = vector_mul(a, b, 3);
    printf("%f", result);
    return 0;
}
```

Bài 3: Bài này là phức tạp nhất:

Mảng 2 chiều trong C:

Trong C thì mảng 2 chiều được cấu tạo từ các "mảng của mảng", thế nên một mảng 2 chiều trong C thực chất sẽ được biểu diễn bằng một mảng pointer trỏ đến các mảng khác, việc này khá phức tạp, ở lời giải này mình chọn cách cấp phát động như sau:

Ở đây pointer temp_data sẽ chỉ đến n mảng được cấp phát động có chiều dài l, tức kích thước ma trận là n * l:

Để thực hiện nhân ta chỉ cần thực hiện 3 vòng lặp for để làm.

```c
#include <stdio.h>
#include <stdlib.h>
// Return an array from function is not easy, that is the point of this example
// There is serveral way to do this, by using pointer or create a struct typedef
// This is the example of using a pointer for 2d array.
// To multipy 2 matrix, the row of the first matrix must equal to the column of
// the second matrix, so we use m to represent that number, n to represent first matrix
// column and l for second matrix row
float ** matrix_mul(float a[][3], float b[][2], int m, int n, int l){
    float ** temp_data = (float **) calloc(n,(n)*sizeof(float *)); // This is a pointer to the heap section of memory, I use calloc to allocate n*l data for the
                                                                   // function to return
    for(int i = 0; i < n; i++){
        temp_data[i] = (float *) calloc(l,(l)*sizeof(float));
        for(int j = 0; j < l; j++){
            for(int k = 0; k < m; k++){
                temp_data[i][j] += a[i][k]*b[k][j];             // Matrix multipication
            }
        }
    }
    return temp_data;
}
int main(){
    float a[2][3] = {{1,1,1}, {2,2,2}};
    float b[3][2] = {{4,4}, {5,5}, {6,6}};
    float **result = matrix_mul(a, b, 3, 2, 2);

//  printf("%f", result);

        for(int i = 0; i < 2; i++){
            for(int j = 0; j < 2; j++){
                printf("%f", result[i][j]);
            }

        printf("\n");
}
    free(result);
    return 0;
}
```

Để lấy data vào, ta sử dụng biến float a[][3] và float b[][2], đây cũng là các pointer trỏ đến pointer, chỉ số [3] và [2] là kích cỡ của mảng được con trỏ trỏ tới, cũng là kích thước tối đa của ma trận.

Cuối cùng vì C không có bộ dọn rác (Garbage Collector) như các ngôn ngữ mới, ta cần giải phóng bộ nhớ đã dùng bằng câu lệnh free(result)

Cách 2: Sử dụng struct: Cách này đơn giản hơn vì ta xây dựng một kiểu dữ liệu mới, dễ dàng xử lý hơn:

```c
#include <stdio.h>
#include <stdlib.h>
// Return an array from function is not easy, that is the point of this example
// There is serveral way to do this, by using pointer or create a struct typedef
// This is the example of using a pointer for 2d array.
// To multipy 2 matrix, the row of the first matrix must equal to the column of
// the second matrix, so we use m to represent that number, n to represent first matrix
// column and k for second matrix row

struct Matrix{
    int row, col;
    float * data;
};
Matrix matrix_mul(Matrix a, Matrix b){
    Matrix temp;
    temp.data = (float *) calloc(b.row*a.col,(a.col*b.row)*sizeof(float));
    temp.row = b.row;
    temp.col = a.col;
    for(int i = 0; i < a.col; i++){
        for(int j = 0; j < b.row; j++){
            for(int k = 0; k < a.row; k++){
                temp.data[i*b.row + j] += a.data[i*a.row + k]*b.data[k*b.row +j];
            }
        }
    }

    return temp;
}
int main(){

    Matrix a;
    Matrix b;
    Matrix c;
    float temp1[6] = {1,1,1,2,2,2};
    float temp2[6] = {4,4,5,5,6,6};
    a.row = 3;
    a.col = 2;
    a.data = temp1;
    b.row = 2;
    b.col = 3;
    b.data = temp2;
//  printf("%f", result);
```

Bằng cách sử dụng 1 mảng 1 chiều ta có thể cài đặt được tính chất của mảng 2 chiều, tuy nhiên thú vị là cách này cũng cho memory layout giống với mảng 2 chiều thực sự:

https://stackoverflow.com/questions/2565039/how-are-multi-dimensional-arrays-formatted-in-memory

Cách 3: Sử dụng tham chiếu đến mảng 2 chiều (cách này các bạn làm rất nhiều :v)

Cách này thì cũng không có gì nhiều để nói, giống bài 2:

```c
#include <stdio.h>
#include <stdlib.h>
// Return an array from function is not easy, that is the point of this example
// There is serveral way to do this, by using pointer or create a struct typedef
// This is the example of using a pointer for 2d array.
// To multipy 2 matrix, the row of the first matrix must equal to the column of
// the second matrix, so we use m to represent that number, n to represent first matrix
// column and l for second matrix row
void matrix_mul(float a[][3], float b[][2], float c[][2], int m, int n, int l){
    float ** temp_data = (float **) calloc(n,(n)*sizeof(float *)); // This is a pointer to the heap section of memory, I use calloc to allocate n*l data for the
                                                                   // function to return
    for(int i = 0; i < n; i++){
        temp_data[i] = (float *) calloc(l,(l)*sizeof(float));
        for(int j = 0; j < l; j++){
            for(int k = 0; k < m; k++){
                c[i][j] += a[i][k]*b[k][j];              // Matrix multipication
            }
        }
    }

}
int main(){
    float a[2][3] = {{1,1,1}, {2,2,2}};
    float b[3][2] = {{4,4}, {5,5}, {6,6}};
    float c[2][2];
    matrix_mul(a, b,c, 3, 2, 2);

//  printf("%f", result);

        for(int i = 0; i < 2; i++){
            for(int j = 0; j < 2; j++){
                printf("%f",c[i][j]);
            }

      printf("\n");
}
    return 0;
}
```