M03S02 [Multi Dimensional Arrays, Pointers and Function]: Practice Drill

CS110: Computing Lab
Department of CSE, IIT Guwahati
Jan-May 2018

Objective of the Stage

In previous stage 1d-Array and Strings, which is array of characters, are covered. Primary objective of this drill is to make you practice with multi-dimensional arrays, pointers and non-recursive functions.

Reference Lecture Slides

- 1st Feb Lecture [Array and Pointer I] [`http://jatinga.iitg.ernet.in/~cs101/2018/Lec12feb1.pdf`]
- 6th-7th Feb Lecture [Pointer] [`http://jatinga.iitg.ernet.in/~cs101/2018/Lec13feb6-7.pdf`]
- 14th Feb Lecture [Array and Pointer II] [`http://jatinga.iitg.ernet.in/~cs101/2018/Lec14feb8.pdf`]
- 15th Feb Lecture [Functions I(Ignore Recursive Function)] [`http://jatinga.iitg.ernet.in/~cs101/2018/Lec17feb15_16.pdf`]

Learning Goal

After doing this practice drill and working on assessment problems of this stage, we expect you to be comfortable with

- basic understanding of multi dimensional array, pointers and functios
- declaration, initialization of a multi-dimensional array
- accessing individual array element and performing some basic operations
- declaration, initialization of a pointers

- performing some basic operations with pointers
- function declaration, function definition and function calling
- passing parameters and returning a value from function

Multi-Dimensional Arrays

The simplest and most used form of multi-dimensional array is 2d Array, which is essentially, an array of one-dimensional arrays. Keeping this basic concept in mind, one can imagine higher dimensional arrays. Consider, you need to create a *Matrix* of 5x5, based on your previous knowledge, either you would have declared 25 variable of same type, which is very very tedious or you would have created 5 1d arrays, again that is bit tedious. Using 2d Array, it can be done using just 1 variable `int matrix[5][5]`

Two-Dimensional Arrays

A two-dimensional array is an arrays. The general form of a two-dimensional array declaration is:

type variable_name[size][size],

where type is base type of the array (e.g. `int`) determining the data type of each element in the array; first size indicates the `number of arrays or rows`; second size indicates the `number of elements that each array can contain or columns`; and *variable_name* is the name of the array.

The size of an array must be an `int` constant or a constant expression.

Examples:
`int matrix[10][5]` indicates *matrix* of type `int` of dimension 10 x 5 i.e. 10 rows and 5 columns;
`char names[10][40]` indicates *names* is a list of 10 names where each name is an array of type `character` which can hold 40 characters;

Initialization

Arrays can be initialized during declaration as shown below:

- `int A[3][3] = {{1,4,5},{2,3,7},{6,8,9}}`
- `char A[5][10] = {"Tom", "Jerry", "Topsy", "Sylvester", "Tweety"}`

Taking input from the user:

```
float x[5][5];
for(int i = 0; i < 5; ++i)
  for(int j = 0; j < 5; ++j)
     scanf("%f",&n[i]ij]);
```

printing 2d-Array:

```
  float x[5][5];
  for(int i = 0; i < 5; ++i)
    for(int j = 0; j < 5; ++j)
      printf("%f",n[i]ij]);
```

Practice Exercises on multi-dimensional arrays

Now, we ask you to do some practice problems given below. This time we are not providing you the code for these problems but ask you to try first by yourself and if you face difficulty, take help of any textbook or TA during the lab session.

1. Write a program in C to check whether the matrix is Diagonal Matrix or not. Use any random number generator to generate 1s and 0s, and use these numbers to initialize array.
2. Write a program in C to perform matrix multiplication.
3. Write a program in C to perform arithmetic operations(add and sub) in 3d Matrix.
4. Write a program in C to accept 5 names starting from different alphabet from user. Sort and display the names.

Pointers

In C, a **pointer** is a variable that contains address of other variables.
Variable are stored in memory. And memory can be considered as a very large array, where every location has an address, an integer, just like an array index.
Memory index in C is known as Pointer.

Two useful operators are :

1. & (`'address of'`) operator: Returns the address of its argument, argument is *Variable Name*.
2. * (`'dereference'`) operator: Returns value stored at a given memory address, that must be a *pointer*.

```
/* Source code to demonstrate, handling of pointers in C program */
#include <stdio.h>
int main(){
   int* pc;
   int c;
   c=22;
   printf("Address of c:%u\n",&c);
```

```c
   printf("Value of c:%d\n\n",c);
   pc=&c;
   printf("Address of pointer pc:%u\n",pc);
   printf("Content of pointer pc:%d\n\n",*pc);
   c=11;
   printf("Address of pointer pc:%u\n",pc);
   printf("Content of pointer pc:%d\n\n",*pc);
   *pc=2;
   printf("Address of c:%u\n",&c);
   printf("Value of c:%d\n\n",c);
   return 0;
}
```

Pointer Arithmetic

Arithmetic Operations allowed with pointers are :

1. increment (++)
2. decrement (–)
3. integer addition
4. integer subtraction

Pointers and Arrays

Pointers and Arrays are closely related. In fact an array variable is actually pointer to the first element of array.
Equivalent Array and Pointer Notation :

1. a[0] is same as *a
2. a[1] is same as *(a+1)
3. a[2] is same as *(a+2), and so on...

If pa is pointer to a particular element of an array,

1. pa + 1 will point to next element of the array
2. pa + i will point to ith element after pa
3. pa - i will point to ith element before pa

Consider the following code :

```c
#include <stdio.h>

int main()
{
   int data[5], i;
   printf("Enter elements: ");

   for(i = 0; i < 5; ++i)
     scanf("%d", data + i);

   printf("You entered: \n");
   for(i = 0; i < 5; ++i)
      printf("%d\n", *(data + i));

   return 0;
}
```

Practice Problems

Now is the time for some practice problems.

1. Write a program in C to demonstrate the use of &(address of) and *(value at address) operator.
2. Write a program in C to perform arithmetic operations two numbers using pointers.
3. Write a program in C to compute the sum of all elements in an array using pointers.
4. Write a program in C to find the maximum number between two numbers using a pointer.

Functions

A function is a set of statements that take inputs, do some specific computation and produces output. The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.

```c
#include <stdio.h>
void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

Example:

```
#include <stdio.h>

// An example function that takes two parameters 'x' and 'y'
// as input and returns max of two input numbers
int max(int x, int y)
{
    if (x > y)
      return x;
    else
      return y;
}

// main function that doesn't receive any parameter and
// returns integer.
int main(void)
{
    int a = 10, b = 20;
    // Calling above function to find max of 'a' and 'b'
    int m = max(a, b);
    printf("m is %d", m);
    return 0;
}
```

Function Declaration

Function declaration tells compiler about number of parameters function takes, data-types of parameters and return type of function. Putting parameter names in function declaration is optional in function declaration, but it is necessary to put them in definition. Below are example of function declarations. (parameter names are not there in below declarations)

```
// A function that takes two integers as parameters
```

```c
// and returns an integer
int max(int, int);

// A function that takes a char and an int as parameters
// and returns an integer
int fun(char, int);
```

Parameter passing to functions

The parameters passed to function are called **actual parameters**. For example, in the above program 10 and 20 are actual parameters.
The parameters received by function are called **formal parameters**. For example, in the above program x and y are formal parameters.
There are two most popular ways to pass parameters :

1. **Pass by Value:** In this parameter passing method, values of actual parameters are copied to functions formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.

   ```c
   #include <stdio.h>
   void fun(int x)
   {
      x = 30;
   }

   int main(void)
   {
       int x = 20;
       fun(x);
       printf("x = %d", x);
       return 0;
   }
   ```

2. **Pass by References:** Both actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.

   ```c
   # include <stdio.h>
   void fun(int *ptr)
   {
       *ptr = 30;
   }

   int main()
   ```

```
{
  int x = 20;
  fun(&x);
  printf("x = %d", x);

  return 0;
}
```

Practice Exercises on Functions

Now, we ask you to do some practice problems given below.

1. Write a program in C to swap 2 numbers using both pass by value and pass by reference.
2. Write a program in C to convert decimal number to binary number using the function.
3. Write a program in C to check whether a number is a prime number or not using the function.
4. Write a program in C to check armstrong and perfect numbers using the function.