Multidimensional Arrays

Prepared by: RNC Recario
rncrecario@gmail.com
Institute of Computer Science UPLB
Dec 2011

OBJECTIVES

At the end of the laboratory session, the student is expected to

- Differentiate a simple array from a multidimensional array
- Perform array data manipulation and computation
- Create a program that solves a problem using multidimensional arrays.

ARRAYS

Suppose I ask you to create a program to add 200 numbers for you to be able to obtain their sum. A typical student would solve the problem using one of the basic two solutions (or variations of the two). Can you think what those two solutions are?

The first solution involves using a counter variable which determines how many numbers were "collected" from the user and accumulates them in another variable let us say sum. The loop terminates when there are already 200 numbers. A sample implementation of such approach goes like this:

```
int ctr=0, sum=0, num=0;

while(ctr<200){
     printf("Enter a number: ");
     scanf("%d", &num);
     sum+=num;
    ctr++;
}
```

The above example is optimal and efficient as you can adjust the condition to whatever number of items you want to add.

Still, another solution "most" novice programmers would do is to declare 200 individual variables for getting the input and another variable say sum to accumulate the sum of the 200 declared variables. The approach would look something like this:

```
int a1, a2, a3, a4, a5, a6, a7,… /*continue the pattern*/, a199, a200,
sum;

printf("Enter a number: ");
scanf("%d", &a1);
printf("Enter a number: ");
scanf("%d", &a2);
/*repeat pattern*/
printf("Enter a number: ");
scanf("%d", &a200);
sum = a1 + a2 + a3 + … /*repeat pattern*/ + a199 + a200;
```

Now, suppose that from computing the sum, I also want to know the most common number in the set (technically, the number is the mode or the number with the most number of occurrences).

What can you say about the first and second approach? How about their advantages and disadvantages?

The problem shown above is an example of a problem that can solved easily by an array. Loosely speaking, an array is a collection of items with a definite number (that is, they are finite). In a stricter definition, an array is a collection of finite items that has the same data types. Thus, from our basic data types int, char, double and float, we can create arrays of int, char, double, and float.

An array is declared using the following syntax:

<data type> <variable name>[size1][size2][size3]…[sizen];

where

<data type>	is the data type of the declared array
<variable name>	is the variable name for the array
[size1][size2][size3]…[sizen]	is the dimensions of the array

If for example that we declare something like this:

```
int scores[20];
```

We mean that we are declaring 20 integer-variables accessible using score. The 1st, 2nd, 3rd up to the 20th item in the list can be accessed using the index (the value inside the `[]`). For example, the first item is accessed using `scores[0]` while the last item (the 20th item) can be accessed using `scores[19]`. Unlike programming languages like Pascal, the index cannot be adjusted. Thus, for an array of size n, the indexes available are from `0` to `n-1`.

Additionally, `scores[20]` is a 1D array of twenty items. For arrays with greater dimensions, the number of items or cells can be computed by multiplying the values of the dimensions within the declaration. Example, suppose we have the following declarations:

```
double grades[4][120];
double granule[4][6][20];
```

The first array `grades` has 480 items while `granule` has 480 items. Now, what is the difference between the two?

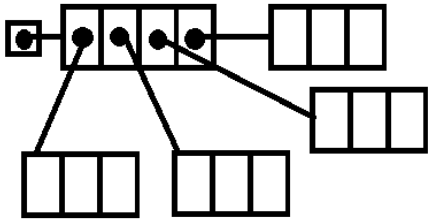
The difference is how we visualize them in the memory. Let's show a simple example. Assume the following declarations:

```
int cake1[12];
int cake2[4][3];
int cake3[2][2][3];
```

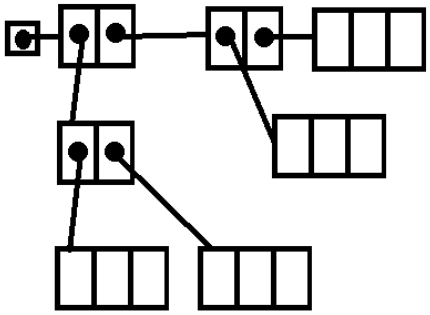
So, again, what is the difference among the three arrays?

The array cake1 can be visualized as this one:

The array cake2 can be visualized as this one:



Still, the cake3 array can be visualized as this one:



What do you observe? They are actually represented by pointers.

ARRAY EXAMPLES

Here are some codes about arrays.

```
main(){
      int s[4][2];
      int i, j;
      for(i =0; i < =3; i ++)
      {
            printf("\n Enter roll no. and marks");
            scanf("%d %d", &s[i][0], &s[i][1] );
      }
      for(i = 0; i < = 3; i ++)
      printf("\n %d %d", s[i][0], s[i][1]);
}
```

A modular version of the same code above can be generated by the code below. Notice how it is easy and similar to our previous knowledge of parameter passing via pass by value.

Thus, regardless of the dimension, it is *supposed to be* quite easy. ☺

```
void printArray(int s[4][2]){
      int i, j;
      for(i = 0; i < = 3; i ++)
      printf("\n %d %d", s[i][0], s[i][1]);
}

void getInput(int s[4][2]){
      int i, j;
      for(i =0; i < =3; i ++)
      {
            printf("\n Enter roll no. and marks");
            scanf("%d %d", &s[i][0], &s[i][1] );
      }
}

main(){
      int s[4][2];
       getInput(s);
       printArray(s);
}
```

Notice that in the actual parameter, what is being passed is not `s[4][2]` but `s` which is a pointer to the memory location of the 4x2 integer matrix.

In order to completely make it dynamic, we must be knowledgeable about the two important functions `malloc()` and `free()` from the `stdlib.h` library.

`malloc()` – allows the user to allocate a memory space depending on the size the user has set.
`free()` – frees the allotted memory space from usage.

The details of how they will be used will no longer be discussed here but will be discussed by your instructor during your laboratory session.

Exercise 4: Multidimensional Arrays

Prepared by: RNC Recario
rncrecario@gmail.com
Institute of Computer Science UPLB
Dec 2011

Download the necessary file `recarioexer4.zip` from our Google site. The zip file is composed of two files: a C file and an H file. You are required to edit the H file in order to make the program work. However, for both files, DO NOT FORGET to document your work.

The Problem:
The files address the matrix addition problem. Suppose we have a matrix A and a matrix B with the same size, say n x m. The matrix addition of A and B, let us say matrix C is computed by

$C_{i,j} = A_{i,j} + B_{i,j}$

where i and j are the indexes of the matrix. You are not allowed to add an additional function in the H file unless there is indeed a missing function (bring this to the attention of your instructor). The function prototype together with the function bodies will give you the idea of is needed to be furnished in the program.

Filename: `cmsc21u<section>lexer4<surname>.c`
Example: `cmsc21u0lexer4recario.c`

Send the COMPRESSED exercise (name it as `<surname>exer4`) to `exercisereceiver@gmail.com` with a subject heading `cmsc21 u<section>l exer4 <surname>.`