J. Appl. Environ. Biol. Sci., 4(7S)22-32, 2014

ISSN: 2090-4274
Journal of Applied Environmental and Biological Sciences
www.textroad.com

What Should Be Taught about Arrays in CS2?

Muhammad Shoaib Farooq[1,2], Sher Afzal Khan[2], Aqsa Ali[1], Adnan Abid[1]

[1]Department of Computer Science, University of Management and Technology, Lahore, Pakistan
[2]Department of Computer Science, Abdul Wali Khan University, Mardan, Pakistan

Received: September 1, 2014
Accepted: November 13, 2014

ABSTRACT

In this article we have presented a thorough discussion on an important topic, the array which is taught in the fundamental courses in computer programming. To this end, we have presented a taxonomy of arrays based on the following four main topics: i) Memory representations; ii) Mapping Functions; iii) Subscript type; and iv) Abstract data types. We also suggest a flow of teaching these topics to the students. We believe that this effort will be useful for the instructors to plan their courses, and will be a good source of learning in general.

KEYWORDS: Array implementation, teach array, arrays mapping, teaching array, abstract data type, teach array CS1, CS2

1 INTRODUCTION

The array is a collection of consecutive elements of the same data type 12345[19]. It is a basic data structure which is introduced to the students in CS1 course, where the idea is to impart the major concepts pertaining to this topic to the students of CS1. The importance of this data structure can be highlighted by the fact that it is used to solve many problems in computer science. Therefore, strong concepts of arrays should be taught by instructors in CS1, whereas detailed implementation and advanced concepts related to the topic should be covered in the CS2 course. The array is a data structure in which a mapping from one finite set (Array indices) to another finite set (array locations) 12345. Every element of the array can be accessed by its index. For example an array of 5 integer variables 10, 20, 30, 40, 50, with indexes 0 through 4, as shown in Figure 1.

Index	0	1	2	3	4
Data	10	20	30	40	50

Figure. 2. Mapping of integer variables onto the indexes of array

Traditionally, concepts of array are introduced in the computer science undergraduate program curricula in the first course related to the computer programming, generally known as, CS1 6. At this stage basic functionality of arrays is introduced to the students such as array declaration, array initialization, getting the contents of the array from console, access any location of an array, assigning some value to an index of the array. The most common types of problems solved in CS1 course are: searching an element in the array, sort the data in an array, sum, and average of the contents of an array etc. The major focus of the instructor is on problem solving through arrays, but technical issues of arrays are not covered in CS1 due to time constraint, course outline and course learning outcomes. Although, these contents give the student a general idea about the data structure and its usage, yet, they are not enough for students to grasp deeper technical concepts of arrays in a comprehensive manner. Therefore, this topic is usually carried forward in CS2 course, and more relevant and advanced concepts related to arrays are recommended to be covered in the CS2 course 6.

In this research, we present a taxonomy of arrays as shown in the Figure 2, based on four major perspectives, including: i) Memory representations; ii) Mapping Functions; iii) Subscript type; and iv) Abstract data types. We have further explained these high level topics into specialized sub-topics so as to make it convenient for the course instructor to define her course outline while adding the topics related to

*** Corresponding Author:** Muhammad Shoaib Farooq, Department of Computer Science, University of Management and Technology, Lahore, Pakistan. {Shoaib.farooq, aqsa.ali, adnan.abid}@umt.edu.pk}

concepts of array in CS2. These contents should be covered in CS2 by instructors because these are essential concepts for a computer science student in order to solve many computing problems.

The rest of the paper is organized as follows: we present the related work in Section 2. In Section 3, we have discussed and elaborated each major perspective, and its relevant sub-topics. At the end of this section, we have suggested a sequence in which these topics should be covered. Finally, conclusion and future directions have been discussed in Section 4.

2 RELATED WORK

First time array introduced by Konrad Zuse's Plankalkül in 194511011. The concrete implementation of array found in FORTRAN I compiler developed by Backus [5] and ALGOL 60 compiler. All mainstream programming languages follow the same concept introduced by FORTRAN I and ALGOL. Ventura et al 6proposes that CS2 is an ideal place for introducing array. They focus on developing abstract data type using an array and also highlights the importance of library collection based on arrays. ACM curriculum 2001, 2008 and 2013141516proposes to focus array topics in CS1 and CS2 course. Kent 9 introduces general array implementation of association list (i.e. associative arrays). Lang at el. 8discussed the importance of ADTs and proposes that it should be introduced in CS2 course at the beginning. Farooq at el. [17] proposes comprehensive framework for the evaluation of the first programming languages and reports safety issues of arrays.

3 Classification of arrays

In following figure taxonomy of arrays is described. There are five classifications of arrays based on subscript binding and array categories.

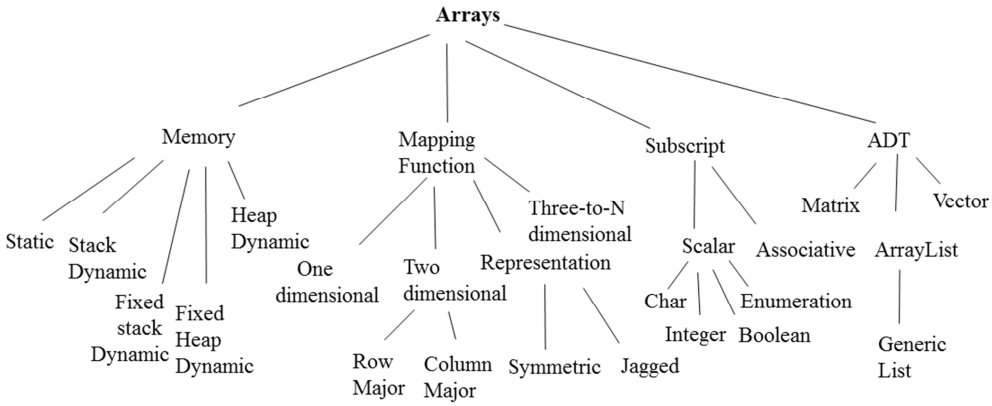

Figure. 2. Array topics covered in CS2

3.1 Memory

This section describes the classification of arrays according to memory, mapping function, subscript, abstract data types (ADT). There are five types of memory classified as static, stack dynamic, fixed stack dynamic, heap dynamic, and fixed heap dynamic. In mapping function one dimensional and two dimensional arrays i) row major ii) column major and their representation (i.e. jagged and symmetric) were described with examples. In subscript there are two types scalar and associative arrays, scalar further divided into four types i) char, ii) enumeration iii) Boolean iv) integer. In ADT ArrayList, vector and generic ArrayList were presented.

Static arrays. Static arrays are those arrays whose sizes are known or defined at compile time and should not be deallocated until the whole program expires 12345Due to fix the size of the array there is no need for allocation and deallocation required at run time, and also enhances program performance. Consequently, static arrays monopolies memory during processing of program life time, it's a matter of time/space tradeoff.

For example, C++ supports these type of arrays using syntax,*static int a[10].* Here Array declared as static occupies memory before execution of the program.

Fixed stack dynamic arrays. A fixed dynamic array is one in which size of the array or subscript ranges are fixed (bounded at compile time) and allocation of memory is done on elaboration of declaration statement at execution time 12345. Life time exists between block and both subscript range and allocated size remain fixed during life time of array. Size must be known at compile time is a major disadvantage of these types of array and on the other hand, space efficiency is a major advantage (Code Listing 1 line 2). Here (Code Listing 1) array *a* created in memory upon elaboration of declaration statement at line 2 and resides in memory up to line 4 and deallocated from memory after expiry of function block at line 5.

Code Listing 1: Fixed Stack Dynamic arrays in C++

```
1.  voidarray_function(){
2.  int a[10];
3.  for (inti=0;i<10;i++)
4.  a[i]=i;
5.  }
```

Stack dynamic arrays. A stack dynamic array is one in which the size and allocation of memory storage done at execution time 12345Life time exists between block and both subscript range and allocated size remain fixed during the life time of the array. So we have no need to know about the size of array at compile time, but it should be known before use at execution time. Flexibility of programming is a major advantage of these types of arrays. C/C++, C#, Java do not support these types of arrays.

Code Listing 2: Stack Dynamic arrays in ADA

```
1.get(index);
2.declare
3.myarray:array(1..index) of integer;
4.begin
5.  end;
```

ADA support stack dynamic array in a local procedure or block. Here (Code Listing 2) user's input at line 1 and deallocation of memory at line 5.

Fixed heap dynamic arrays. A fixed heap dynamic array is similar to fixed stack dynamic arrays except the memory used is heap and both the subscript range and allocation is done at execution time12345. After allocation at execution time subscript range and allocation is fixed during the life time of the program (Code Listing 3). Here (Code Listing 3) *a*'s range is statically bound & its storage bounding is dynamic. C/C++, C#, Java support these types of arrays. C use *malloc* and *free* library functions, while C++ use *new* and *delete* keywords. C# and Java (Code Listing 4) use *new* keyword and deallocation is a responsibility of the programmer.

Code Listing 3: Fixed heap dynamic arrays in C++

```
1.  int *a;
2.    a=new int[20];
3.    a[1]=13;
4.  delete []a;
```

Code Listing 4: Fixed heap dynamic arrays in JAVA

```
1. int size = 10;
2. int a[ ]  = newint [size];
```

Heap dynamic arrays. In heap dynamic arrays, allocation and deallocation of storage can grow or shrink and size can be changed many numbers of times during the whole lifetime of the array12345. Flexibility of programming is a major advantage of these types of arrays. Perl, Java, C# support these types of arrays. Here (Code listing 5) create an array of five numbers at line 1and then add more elements at line 2. At the end array empty list assigned to array at line 3.

Code Listing 5: Heap Dynamic arrays in Perl

```
1. @list = (1,2,5,7,9);
2. Push(@list, 13, 15);
3. @list= ( );
```

3.2 Mapping functions

Arrays are linear data structures either one or many dimensions. In order to access a specific location in the array a mapping function is used 1234512. Mapping function can be one dimensional, two dimensional and N-dimensional. There are two types of mapping functions for two to N dimensional array i) row major and ii) column major, support two views(i.e. logical and physical). Physically all arrays are mapped as flattened and mapping function is used to support logically transparent access for multidimensional arrays.

Multidimensional arrays are flattened arrays based on row and column major ordering. Most the main stream programming languages support either row major or column major.

One dimensional. To calculate the index of one dimensional array, add *the base address* with *index* and multiply size of *the data type*. e.g. *a [2] =10;* here base address is *a* (which is starting index of array) and *2* is index and mapping is defined as *Baseaddress + index*Sizeoftype.*

Two dimensional.

Column Major. Column major is a method of flattening array column wise 123457. Column major supported by Pascal, FORTRAN and COBOL. To calculate an index of two dimensional array indexes, mapping function is defined as *BaseAddress+ (j*n+i) *sizeofType.*

i=row index
j=col index
n= size of total rows in matrix

Example. Suppose an array *a* consist of two rows and three columns declared in C++. Here we assume base address is 100.

int a[2][3];

Array *a* can be seen as a two dimensional matrix (Logical view) but actually in the memory array is flattened (Physical view) as shown in Figure 3, and Figure 4.

10	20	30
40	50	60

Figure. 3. Logical view

10	40	20	50	30	60

Figure. 4. Physical view

In a typical assignment statement a[0][0]=10 , value 10 will be assigned at 100 address in memory using mapping function 100+(0*2+0)* 4=100.

In another typical assignment statement a[1][0]=40, value 40 will be assigned 104 address in memory using mapping function 104+(0*2+1)*4= 108.

Row Major. Row major is a method of flattening array row wise. Row major supported by C, C++, Java and other mainstream programming languages 12345. To calculate the address of two dimensional array indexes, mapping function of row major is defined as:

BaseAddress+ (i*n+j) *sizeofType

i=row index

j=col index

n= size of total columns in matrix

Example. Suppose an array *a* consist of two rows and three columns declared in C++. Here we assume baseAddress is 100.

int a[2][3];

Array *a* can be seen as a two dimensional matrix (Logical view) but actually in the memory array is flattened (Physical view) as shown in Figure 5 and Figure 6:

10	20	30
40	50	60

Figure. 5. Logical view

10	20	30	40	50	60

Figure. 6. Physical view

In a typical assignment statement *a[0][0]=10 ,* the value *10* will be assigned at *100* address in memory using mapping function*100+(0*3+0)* 4=100*

In another typical assignment statement *a[1][0]=,*value *40* will be assigned *112* address in memory using mapping function*100+(1*3+0)*4=112*.

Three-to-N dimensional array. The Three-to-N dimensional array are also called cube, they represents multi-arrays an array of arrays. Mapping function of these array based on recursively define two-dimensional mapping function. Generic formula for three-to-N dimensional array are given below:

$A[D_0] [D_1]\ldots[D_{n-1}]$

Here A is an array of n dimensions. For row major representation, slice along the 1st dimension to get an array of N-1 dimensions, continue until you are left with one dimension only. For offset of $a[d_0] [d_1] \ldots[d_{n-1}]$ generic mapping function is written as:

$BaseAddress+(((d_0*D_1 + d_1)*D_2 + d_2)*D_3 + \ldots) + d_{n-1}) * sizeoftype$

Example 1. Suppose a three dimensional array *a* consist of 2,3,4 dimensions declared in C++. Here we assume base address is 100 and size of *int* data type is 4.

int a[2][3] [4];

$D_0 D_1$ D2

a[1] [2] [1]=10;

d_0 d_1 d2

Mapping Function: *Base Address + [(do*D1+d1)*D2+d2]*4*
Final value according to function:*100+ [(1*3+2)*4+1]*4=184*

Four dimensional. As usual can alsocalculate the address of four dimensional array indexes, the mapping function is defined as
*Base Address+ [((d0*D1+d1)*D2+d2)*D3+d3]*size of type*

Example 2. Suppose a four dimensional array *a* consist of 2,3,4,5 dimensions declared in C++. Here we assume base Address is 100 and size of *int* data type is 4.

int a[4][5] [6] [7];
$D_0 D_1$D2 D3

a[2] [1] [3] [2]=20;
d_0 d_1d2 d3

Mapping Function:

*Base Address + [(d0*D1+d1)*D2+d2*D3+d3]*4 (8)*

Final value according to function:*100+ [((2*5+1)*6+3)*7+2]*4=2040*

It is pertinent to mention here that the memory representation of arrays can be generalized to any dimensions.

Two dimensional representation. Two Dimensional array represented in two forms i) jagged and ii) symmetric.

Jagged. Jagged (irregular) array has rows with varying number of elements (i.e. columns). Java supports these types of array as shown in code listing 6. The code listing 6 generates a jagged array as shown in fig 7 and code listing 7 shows same jagged array initialization in C++.

Code Listing 6: Jagged Array in Java

```
1.  int jagged[][]=new int [3][]
2.  jagged[0]=new int[2]
3.  jagged[1]=new int[3]
4.  jagged[2]=new int[4]
5.  int k=0;
6.  for(inti=0;i<3;i++)
7.  for(int j=0;j<i+1;j++){
8.  jagged[i][j]=k;
9.  k++;
10. }
```

0	1		
2	3	4	
5	6	7	8

Figure. 7. Jagged Matrix

Code Listing 7: Jagged Array in C++

```
1.int *a[3]
2.  jagged[0]=new int[2]
3.  jagged[1]=new int[3]
4.  jagged[2]=new int[4]
5.int k=0;
6.for(inti=0;i<3;i++)
7.for(int j=0;j<i+1;j++){
8.      jagged[i][j]=k;
9.k++;
10.}
```

Symmetric. It is a multi-dimensioned array in which all rows and columns have the same number of elements as shown in fig 8.

2	**5**
3	7

Figure. 8. Symmetric Matrix

3.3 Classification using subscript type.

Normally access of arrays can be written as array Name *[index]* in most of the languages and subscript(index) type is integer, but in general, basically there are three kinds of subscript types i.e. Scalar and associative.

Scalar Type

Any type that has predictable predecessor and successor 12345. For example: integer is scalar type, predecessor of 4 is 3 and successor is 5. Primitive data types except real, float and double are belongs to scalar. For example: In C++, short, int, unsigned int, long, char, Boolean are scalar types. Indexes never are a float value, because if we take range from 2 to 3 then there are many infinite numbers between these two numbers.

Integer. Most common subscript type supported by many mainstream programming languages. C/C++, Java, C# support only integer subscript.

Character. Character data type can also be used for array subscripts. For example: Pascal support character as subscript data type (Code Listing 8).

Code Listing 8: Charter type array subscript in Pascal

```
1.  Program my Array;
2.  var
3.a:array['a'..'d'] of integer;
4.  begin
5.   a['a']=10;
6.   a['b']=20;
7.   a['c']=30;
8.   a['d']=40;
9.end.
```

Boolean. Boolean is also a scalar type and hence can be used as array subscript. For example: Pascal support character as subscript data type (Code Listing 9).

Code Listing 9: Charter type array subscript in Pascal

```
1.  Program my array;
2.  var
3.  a:array[false..true] of integer;
4.  begin
5.    a[false]=10;
6.    a[true]=20;
7.end.
```

Enumeration. Enumeration are user defined named constants 12345. It is used to enhance the readability of programs. For example enumeration of days declared as:

enum day= {Sun, mon, tue, wed, thr, fri, sat}, Compiler implicitly assigned the integer value for each enumeration starting from 0 to 6. Code Listing 10 shows access of array with defined enumerations.

Code Listing 10: Enumeration in C++

```
1.  enum day={sun, mon, tue, wed, thr, fri, sat};
2.  void main( ) {
3.int a[7];
4.   a[sun]=10;  // assign 10 at location a[0]
5.   a[mon]=20; //assign 20 at location a[1]
6.   cout<<a[sun]<<a[mon]; //output  10     20
7.}
```

Associative arrays. Associative arrays are collection of unordered locations that are indexed on keys also called hashes 12345. Index for this array is called key, and its type called keytype. The keytype of these arrays can be scalar or string; first time introduced in Perl and shows an implementation of hash table. These arrays are declared by placing keytype within array subscript brackets. E.g. *b[''hiiii'']=3*. In this assignment statement, 3 placed at value associated with key ''hiiii''. PHP, Python and Rubi support these types of arrays. Java, C++ and C# support associative arrays with library classes.

3.4 ADT (Abstract data types).

ADT is an abstract data type in which data and related operations are black boxed 12345. Array can be implemented as ADT with name Array List. The Array List is a collection of elements with all related functions such as sort, copy, clone, search, traverse etc.

Matrix. A two dimensional ADT with row and columns defined by user. Major operations are add, multiply, subtract, inverse, transpose etc.

Code Listing 11: Matrix ADT in C++

```
1. #include <iostream>
2. #include <string>
3.using namespacestd;
4.class Matrix{
5.public :
6.  Matrix() { }
7.  Matrix(int r, int c){
8.   row=r;
9.   col=c;
10.  p=new int[row*col];
11.for (inti=0;i<row*col;i++)
12.     p[i]=0;
13.  }
14.void get(){
15.for (inti=0;i<row;i++)
16.for (int j=0;j<col;j++)
17.cin>>*(p+i*col+j);
18.  }
19.void print(){
20.for (inti=0;i<row;i++){
21.for (int j=0;j<col;j++)
22.cout<<*(p+i*col+j)<<"";
23.   cout<<endl;
24.  }
25  }
26.  Matrix multiply(Matrix b){
27.  Matrix temp(row,b.col);
28.for (inti=0;i<row;i++)
29.for (int j=0;j<b.col;j++)
30.for (int k=0;k<col;k++)
31.temp.p[i*b.col+j]=
temp.p[i*b.col+j]+ p[i*col+k]* b.p[k*b.col+j];
32.return temp;
33.   }
34.private :
35.int *p;
36.int row;
37.int col;
38. };
39.void main(){
40.  Matrix a(2,3);
41.a.get();
42.  Matrix b(3,2);
43.b.get();
44.  Matrix c(2,2);
45.  c=a.multiply(b);
46.c.print();
47. }
```

Matrix logically shows two dimensional array but are based on one dimensional array. For Example, Matrix ADT in two dimensional array in C++ with get, print, multiply shown in Code Listing 11.

Vector. The vector is an abstract data type implemented with array as data structure. Bjarne Stroustrup inventor of C++ propose vector instead of using C-style array 13. The vector is an ADT defined in C++ STL (Standard Template Library) is type safe with array bound checking. C-style array are not type safe with no bound checking [17] [18]. The elements of a vector are stored contiguously. Like all dynamic array

implementations, vectors have low memory usage and good locality of reference and data cache utilization. Size of a vector can change dynamically, while arrays have fixed size. In java Vectors are synchronized; any method that touches the Vector's contents is thread safe. The reserve space can also be created in vector. Here (code listing 12) of vector in C++ assign 20 at 0 to 9 locations and then print ‘‘20’’ ten times.

Code Listing 12: Vectors in C++

```
1.      vector <int> v;
2.      v.assign(10,20);
3.      for(inti=0;i<v.size( );i++)
4.cout<<v[i];
```

ArrayList. Array List is an abstract data type implemented with array as data structure. Major function supported by Array List is add(), get(), is Empty(), remove(), size(), sort() etc. C++ defines list in its STL (standard template library). Java also supports ArrayList in *java. lang* package. C# also support array List as predefined ADT. Students can develop their own ADTs in Object oriented languages. e.g in c++ we can create ADT as shown in code listing 13.

Code Listing 13: ArrayList in C++

```
1.    class List {
2.    public :
3.   List () { }
4.boolisEmpty(){ }
5.boolisFull(){ }
6.intgetUse(){ }
7.int size(){}
8.bool add(int value, int index){ }
9.bool remove(int&value,int index){ }
10.private:
11.int *p;
12.int size;
13.int  use;
14. };
```

Generic Array List. The ArrayList is type depended abstract data type and can be used only for types that are hardcoded in arrayList class. For example in code listing 9, we can use this array list for only integer data type declared at line no 11.

Generic arrayList is type independent ADT. There is no type in class definition, but type should be mentioned by programmer explicitly at declaration time. Compiler generates appropriate code for each given type before execution. Code listing 14 shows generic arrayList in C++.

Code Listing 14: Generic ArrayList in C++

```
1.    template<class T>
2.    class List {
3.    public :
4.   List () { }
5.boolisEmpty(){ }
6.boolisFull(){ }
7.intgetUse(){ }
8.int size(){}
9.bool add(T value, int index){ }
10.bool remove(T &value, int index){ }
11.private:
12.   T *p;
13.int size;
14.int  use;
15. };
```

3.5 Sequence of Instructions

These topics should be taught in the sequence and proposed class hours shown in Table 1.

Table 1. Suggested sequence for teaching related to arrays.

Topic	Sub-Topics (left to right)	Class hours
Memory Representation	Static, Fixed Stack Dynamic, Stack Dynamic, Fixed Heap Dynamic, Heap Dynamic	1hour and 30 minutes
Mapping Function	Single Dimensional, Two Dimensional, Row Major, Column Major, n-dimensional generalization.	1hour and 30 minutes
Subscript Type	Scalar, Associative	30 minutes
Abstract Data Type	Array List, Generic List, Vector, Generic Vector, Matrix	3 hours

The proposed sequence suggests to teach the topic of arrays by starting with the basic concept of memory representation which should be followed by discussion on the mapping function and its different variants. This should be followed by a discussion of possible subscript types. Lastly, the practical implementation of arrays in the form of various ADTs should be discussed which may include array lists, generic list, vector, generic vector, and matrix.

4 Conclusion and future directions

In this work, we have presented a taxonomy of arrays which is based on four major heads that include: i) Memory representations; ii) Mapping Functions; iii) Subscript type; and iv) Abstract data types. We have discussed the main topic of arrays under the aforementioned heads with thorough details with the help of theoretical concepts and code listing. We strongly believe that this will be helpful for the course instructors to design and plan their course, whereas, it will be helpful for the students in the learning of this topic. In future, we intend to work on the other data structures in a similar fashion.

REFERENCES

1. Sebesta, R. W. “Concepts of programming languages 10 edition”, Addison Wesley, 2012.
2. Pratt, T. W., Zelkowitz, M. V., & Gopal, T. V. “Programming languages: design and implementation”. Englewood Cliffs: Prentice-Hall1984; pp. 147-147.
3. Sethi, R. Programming languages: “concepts and constructs”. Addison Wesley Longman Publishing Co., Inc. (1996).
4. Watt, D. A. “Programming language concepts and paradigms”. Englewood Cliffs: Prentice Hall 1990; Vol. 234.
5. Tucker, A. B., Aiken, R. M., Barker, K., Bruce, K. B., Cain, J. T., Conry, S. E., ...& Chairman-Barnes, B. H. Computing curricula 1991: report of the ACM/IEEE-CS Joint curriculum task force ACM1991.
6. Ventura, P., Egert, C., & Decker, A. “Ancestor worship in cs1: on the primacy of arrays”. In Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications2004 October; pp. (68-72). ACM.
7. Burger, K. R. (2003, February). “Teaching two-dimensional array concepts in Java with image processing examples”. In ACM SIGCSE Bulletin Vol. 35; No. 1; pp. 205-209 ACM.
8. Lang, J. E., & Maruyama, R. K. (1989, February). “Teaching the abstract data type in CS2”. In ACM SIGCSE Bulletin (1989, February Vol. 21; No. 1; pp. 71-73.ACM.

9. Kent, M. “A general-arrays implementation of association lists”. ACM SIGAPL APL Quote Quad, (1993); 23(4); 3-5.

10. Giloi, W. K. Konrad Zuse'sPlankalkül: the first high-level,“non von Neumann” programming language. Annals of the History of Computing, IEEE, 1997;19(2); 17-24.

11. Bauer, F. L., &Wössner, H. The “Plankalkül” of Konrad Zuse: a forerunner of today's programming languages. Communications of the ACM, 1972;15(7); 678-685.

12. Schmit, H., & Thomas, D. E. “Array mapping in behavioral synthesis”. In Proceedings of the 8th international symposium on System synthesis 1995, September; pp. 90-95. ACM.

13. Stroustrup, B. “Programming in an undergraduate CS curriculum”. In Proceedings of the 14th Western Canadian Conference on Computing Education 2009, May; pp. 82-89. ACM.

14. Roberts, E., Shackelford, R., LeBlanc, R., & Denning, P. J. Curriculum 2001: Interim report from the ACM/IEEE-CS task force. In ACM SIGCSE Bulletin 1999, March ;Vol. 31; No. 1; pp. 343-344. ACM.

15. Cassel, L., Clements, A., Davies, G., Guzdial, M., McCauley, R., McGettrick, A., ...&Weide, B. W. (2008). Computer science curriculum 2008: An interim revision of CS 2001.

16. Sahami, M., Roach, S., Cuadros-Vargas, E., & Reed, D.Computer science curriculum 2013: reviewing the strawman report from the ACM/IEEE-CS task force. In Proceedings of the 43rd ACM technical symposium on Computer Science Education 2012, February; pp. 3-4. ACM.

17. Farooq, Muhammad Shoaib, Sher Afzal Khan, Farooq Ahmad, Saeed Islam, and Adnan Abid. “An Evaluation Framework and Comparative Analysis of the Widely Used First Programming Languages”. (2014) PloS one 9(2); e88941.

18. Muhammad Shoaib Farooq, Sher Afzal Khan, Adnan Abid“A Framework for the Assessment of a First Programming Language”, Journal of Basic and Applied Scientific Research, (2012); 2(8); 8144-8149.

19. Muhammad Shoaib Farooq, Adnan Abid, Sher Afzal Khan, Muhammad AzharNaeem, Amjad Farooq, Kamran Abid, “A Qualitative Framework for Introducing Programming Language at High School”, Journal of Quality and Technology Management, Punjab University, Pakistan. 2012; 8(2).