**2. DATA STRUCTURE: ARRAY**

* 1. **What is an Array?**
* Array is an ordered set which consist of fixed number of elements of same data type.
* Array is a linear data structure.
* Array is a collection of variables of same types all of which are referred by a common name.
* A specific element of array is accessed by an index.
* Array is classified into two categories: (1) One dimensional (2) Two dimensional.
  1. **Characteristics of Array**

1. An array holds elements that have the same data type.
2. Array elements are stored in subsequent memory locations.
3. Two-dimensional array elements are stored row by row in subsequent memory locations.
4. Array name represents the address of the starting element.
5. Array size should be mentioned in the declaration. Array size must be a constant expression and not a variable.
6. While declaring the 2D array, the number of columns should be specified and it’s a mandatory. Whereas for number of rows there is no such rule.
7. An Array index by default starts from 0 and ends with the index no. (N-1).
8. The size of an Array cannot be changed at Run Time.  
   1. **One dimensional Array:**

* One dimensional array can be declared as

Example:- int a[20].

index

Data type

Array name

* By the above statement we are declaring an array named as **a** and which can store 20 integer elements.
* While declaring array we need to give three piece of information: data type, array name and its size.
* The size specified in [] square brackets (subscript operator). Size must be given when array is declared.
* Array name is an identifier which follows the rules of writing an identifier.
* Each integer takes 2 bytes so this array declaration reserves 40 bytes of memory for the array when the array element is initializing.
* The first elements of array starts at index 0(means a[0]) and index of last element is n-1.
* The starting index for the array i.e. 0 here is known as Lower Bound (LB) and higher index n-1 is known as Upper Bound (UB).
* Number of elements in array is given by the following formula:

**N=UB-LB+1**

**Storage Representation of One Dimensional Array:**

* In one dimensional array the number of memory location is sequentially allocated to the array. So it is also known as sequential list.
* If each element requires one word in memory then n elements array occupies n consecutive words in memory.
* Since the size of an array is fixed it requires fixed number of memory location.
* Representation of an array in memory location is shown below:

|  |
| --- |
| **A[0]** |
| **A[1]** |
| **A[2]** |
| **….** |
| **A[i]** |
|  |
| **A[N-1]** |

**Base Address(L0)** **2000**

**2001**

**2002**

**L0 + C\*(i - B)**

**2020**

* L0 is the starting(Base) address or address of first element in the array.
* C is the number of words allocated to each element
* B is the starting index of an array.
* Thus location(Address) of A[i] in the array is calculated as:

**Address of A [i]=Loc(A[i]) = L0 + C\*(i - B)**

If we want to find the address of A [2] then it is calculated as:

Loc (A [2]) = L0 + C\*(i - B)

= 2000 + 1 \* (2 - 0)

= 2000 + 2

= 2002

* 1. **Operations on One dimensional Array:**

**We can perform operation such as Traversal, Insert, Delete, Search, and Sorting and Find Big or Small Number of Array**

1. **Traversal:** This operation is used to traverse one dimensional array for inserting or displaying elements of one dimensional array.

**Algorithm: TRAVERSE (A, LB, UB)**

* **A is an array**
* **LB is lower limit of an array**
* **UB is Upper limit of an array**
  1. [Initialize] COUNTLB
  2. [Perform Traversal and Increment counter]

While (COUNT ≤ UB)

* + 1. Write A [COUNT] or Read A [COUNT]
    2. COUNTCOUNT+1
  1. [Finished]

Exit

1. **Insertion:** This operation is used to insert an element into one dimensional array.

**Algorithm: INSERT (A, POS, N, VALUE)**

* **A is an array**
* **POS indicates position at which you want to insert element.**
* **N indicates number of elements in an array.**
* **VALUE indicates value(element) to be inserted.**
  1. [Initialize] TEMPN
  2. [Move the elements one position down]

Repeat While (TEMP ≥ POS)

* + 1. A [TEMP + 1] A[TEMP]
    2. TEMPTEMP-1
  1. [Insert element]

A [POS] VALUE

* 1. [Increase size of array]

NN + 1

* 1. [Finished]

Exit

1. **Deletion:** This operation is used to delete an element from one dimensional array.

**Algorithm: DELETE (A, POS, N, VALUE)**

* **A is an array**
* **POS indicates position at which we want to insert element.**
* **N indicates number of elements in an array.**
* **VALUE indicates value(element) to be inserted.**
  1. [Initialization]

TEMPPOS

* 1. [Move the elements one position up]

Repeat While TEMP ≤ N-1

* + 1. A[TEMP] A[TEMP + 1]
    2. TEMPTEMP+1
  1. [Decrease size of array]

NN - 1

* 1. [Finished]

Exit

1. **Searching:** This operation is used to search particular element in one dimensional array.

**Algorithm: SEARCH(A, N, X)**

* **These function searches the list A consist of N elements for the value of X.**
  1. [Initialize search]

I1

K [N+1] X

* 1. [Search the Vector]

Repeat while K [I] ≠ X

II+1

* 1. [Successful Search?]

If I = N+1 then

Write “Unsuccessful Search”

Return 0

Else

Write “Successful Search”

Return 1

1. **Sorting:** This operation is used to sort the elements of array in ascending order or descending order.

**Algorithm: SORT(A,N)**

* **These function sort elements of array A consist of N elements.**
* **PASS denotes the pass index.**
* **LAST denotes the last unsorted element.**
* **EXCHS counts the total number of exchange made during pass.**
  1. [Initialize]

LASTN

* 1. [Loop on pass index]

Repeat thru step 5 for PASS = 1, 2, …., N-1

* 1. [Initialize exchange counter for this pass ]

EXCHS 0

* 1. [Perform pair wise comparisons on unsorted elements ]

Repeat for I = 1, 2… LAST-1

If K[I] > K[I+1] then

K [I] K [I+1]

EXCHS EXCHS+1

* 1. [Exchange made on this pass?]

If EXCHS = 0 then

Write “Mission Complete”

Else

LASTLAST-1

* 1. [Finished]

Return (Minimum number of pass required)

1. **Find max Number from Array:** This operation is used to find the largest element of an array.

**Algorithm: MAX(A, N)**

* **A is an array**
* **N indicates number of elements in an array.**
  1. [Initialize]

MAX A[0]

* 1. [Scan each element in array]

For I = 1 to N

* 1. [Compare Element]

If A [I] > MAX then

MAX A[I]

* 1. [Output Max and Min]

Write MAX

* 1. [Finished]

Exit

* 1. **Two Dimensional Array (Storage Representation)**

**(1)** **Row-major Arrays**

* One method of representing a two dimensional array in memory is the row major order representation.
* In this representation the first row of the array occupies the first set of memory locations reserved for the array, the second row occupies the next set and so on.
* An array consisting of n rows and m columns can be stored sequentially in row major order as :

Row1 A[1,1] A[1,2] A[1,3] A[1,4]……………..A[1,m]

Row 2 A[2,1] A[2,2] A[2,3] A[2,4]…………….A[2,m]

………………………………………………………….

Row n A[n,1] A[n,2] A[n,3] A[n,4]…………….A[n,m]

* **Example:**

A two dimensional array consist of two rows and three columns is stored sequentially in row major order as:

Column 1 Column 2 Column 3

|  |  |  |
| --- | --- | --- |
| A[1][1] | A[1][2] | A[1][3] |
| A[2][1] | A[2][2] | A[2][3] |

Row 1

Row 2

|  |
| --- |
| A[1][1] |
| A[1][2] |
| A[1][2] |
| A[2][1] |
| A[2][2] |
| A[2][3] |

ROW 1

ROW 2

* The address of element A[i, j] can be obtained by evaluating expression:

**Loc (A [i, j]) = L0 + (i - 1) \* m + (j - 1)**

* Where L0 is the address of the first element(Base Address) in the array.

i is the row index.

j is the column index.

m is the no. of columns.

* for example the address of element A[1,2] is calculated as:

A [1, 2] = L0 + (i - 1) \* m + (j - 1)

Here, m=3, n=2, i=1, j=2

=L0 + (1 - 1) \* 3 + (2 - 1)

=L0 + 0 +1

=L0 + 1

**(2) Column-major Arrays**

* One method of representing a two dimensional array in memory is the column major order representation.
* In this representation the first column of the array occupies the first set of memory locations reserved for the array, the second column occupies the next set and so on.
* An array consisting of n rows and m columns can be stored sequentially in column major order as :

Column 1 A[1,1] A[2,1] A[3,1] A[4,1]……………..A[n,1]

Column 2 A[1,2] A[2,2] A[3,2] A[4,2]……………. A[n,2]

………………………………………………………….

Column m A[1,m] A[2,m] A[3,m] A[4,m]………….A[n,m]

* **Example:** A two dimensional array consist of two rows and three columns is stored sequentially in column major order as:

Column 1 Column 2 Column 3

|  |  |  |
| --- | --- | --- |
| A[1][1] | A[1][2] | A[1][3] |
| A[2][1] | A[2][2] | A[2][3] |

Row 1

Row 2

|  |
| --- |
| A[1][1] |
| A[2][1] |
| A[1][2] |
| A[2][2] |
| A[1][3] |
| A[2][3] |

Column 1

Column 2

Column 3

* The address of element A[i, j] can be obtained by evaluating expression:

**Loc (A [i, j]) = L0 + (j - 1) \* n + (i - 1)**

* Where L0 is the address of the first element(Base Address) in the array.

i is the row index

j is the column index

* for example the address of element A[1,2] is calculated as:

A[1,2] = L0 + (j - 1) \* n + (i - 1)

Here, m=3, n=2 , i=1, j=2

=L0 + (2 - 1) \* 2 + (1 - 1)

=L0 + 2 + 0

=L0 + 2

* 1. **Three or Multidimensional Arrays**
* The C also allows the use of three or more dimensional arrays which are known as multi dimensional arrays.
* The two dimensional are declared as follows:

**Data type array\_name[d1][d2][d3]…[dn].**

* If I want to display student result in form of SPI, CPI, CGPA from sem1 to sem 6. So we have to use three dimensional array.
* In three dimensional array first dimensional is student, second dimensional is sem and third dimensional is result.

**Student 3**

|  |  |  |  |
| --- | --- | --- | --- |
| Sem1 |  |  |  |
| Sem2 |  |  |  |
| Sem3 |  |  |  |
| Sem4 |  |  |  |
| Sem5 |  |  |  |
| Sem6 |  |  |  |
|  | SPI | CPI | CGPA |

|  |  |  |  |
| --- | --- | --- | --- |
| Sem1 |  |  |  |
| Sem2 |  |  |  |
| Sem3 |  |  |  |
| Sem4 |  |  |  |
| Sem5 |  |  |  |
| Sem6 |  |  |  |
|  | SPI | CPI | CGPA |

**Student 2**

|  |  |  |  |
| --- | --- | --- | --- |
| Sem1 |  |  |  |
| Sem2 |  |  |  |
| Sem3 |  |  |  |
| Sem4 |  |  |  |
| Sem5 |  |  |  |
| Sem6 |  |  |  |
|  | SPI | CPI | CGPA |

**Student 1**

* 1. **Strings**
* String is a group of characters.
* The storage structure for strings is same as characters as string is a sequence of characters.
* A string is stored in computer memory as series of ASCII codes of the characters in a string.
* Each string in C is ended with the special character called null character and denoted as ‘\0’ or NULL.
* **Character set of string:**

1. Alphabets (a-z, A-Z)
2. Numeric (0-9)
3. Special Character (+,/,\*,$,@,#,..etc)

* For example:

**String “GOOD” is stored as shown in fig**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **‘G’** | **‘O’** | **‘O’** | **‘D’** | **‘\0’** |

* In the above string four characters need five bytes of storage.
* **String Functions:**

1. Computing length of string.
2. Copy one string into another.
3. Reverse String
4. Concate two strings.
5. Compare two strings.
6. Find substring.

**2.8 Array of structure *OR* Structure and Array**

* Many times we have to store information regarding real word entities like employees in a company or details of students in a particular class.
* Employees or the students in this case are entities which cannot be described by the single data item.

For example:

Student is described by its id number, name, address etc.

* Consider the above example, first we need to define the structure for student. It defined as below

**struct Student**

**{**

**int id\_no;**

**char name[20];**

**char address[50];**

**}s;**

* The above declaration contains information of one student but if we want to store information about 120 students then we use concepts of Array of Structure.

**struct Student**

**{**

**int id\_no;**

**char name[20];**

**char address[50];**

**}s[120];**

* C use the dot (.) operator to access the members of the structure.
* Each and every member is accessed by structure variable like s[i].id\_no, s[i].name, s[i].address, where 0 < i < 120.
  1. **Drawbacks of linear Arrays**
* The major drawback of the linear or one dimensional array is the insertion and deletion operation is complicated and time consuming.
* If we declare array with size 50 and we create array of 20 elements so rest of 30 elements are occupies the memory space, which will be west the memory space.
* The size of array must be known first before it declared, otherwise it create overflow or underflow error.
* The array is best when the size of data or number of elements remains fixed.
  1. **Insertion and deletion operation are difficult and time consuming with array.**

**Insertion with Array:**

* Suppose there are N elements in an array and we want to insert an element between first and second element.
* We have to move last N-1 elements down

**Deletion with Array:**

* Suppose there are N elements in an array and we want to delete an element from array.
* We have to move the elements up to take vacant space after deleting element.

Thus insertion and deletion operation are very difficult and time consuming with array.

* 1. **Pointer and Array**
* Pointer and array are related with each other.
* The name of array itself is address of array i.e. address of first element (a=&a[0]).

For example: int a[5];

|  |  |
| --- | --- |
| \*(a+0) | a |
| \*(a+1) | a+1 |
| \*(a+2) | a+2 |
| \*(a+3) | a+3 |
| \*(a+4) | a+4 |

|  |  |
| --- | --- |
| a[0] | a |
| a[1] |  |
| a[2] |  |
| a[3] |  |
| a[4] |  |

* As a is a first element of array and it also defines base address.
* The only limitation to use array name a as a pointer is that it cannot be modified because array name is address of first element and is always constant.
* So here subscript of array(a[i]) is internally converted into \*(a+i).
* So a[0]=\*(a+0),a[1]=\*(a+1)….etc. here array adds **base address+subscript\*size of data type.**
* If base address is 2000 then address of a[2]=2000+2\*2=2004.
  1. **Pointers and strings**
* A pointer which pointing to an array which content is string, is known as pointer to array of strings.
* **Example: w.a.p to find reverse string of given string.**

#include <stdio.h>  
#include <string.h>  
**void**main()  
{  
   **char**str1[] = "Pointers are fun and hard to use";  
   **char**str2[80], \*p1, \*p2;  
   /\* make p point to end of str1 \*/  
   p1 = str1 + strlen(str1) - 1;  
   p2 = str2;  
   **while**(p1 >= str1)  
     \*p2++ = \*p1--;  
   /\* null terminate str2 \*/  
   \*p2 = '\0';  
   printf("%s %s", str1, str2);  
 getch();

}

* **Inspirational Quotes:**

Concentrate all your thoughts upon the work at hand. The sun’s rays do not burn until brought to a focus.