Data Structure Using C & C++

Class- BCA IIIrd Semester

Dr. Dharm Raj Singh

Assistant Professor, (HOD)

Department of Computer Application

JagatpurP. G. College, Varanasi

Mobile No. 9452070368, 7275887513

Email- dharmrajsingh67@yahoo.com

Outline

1. MODULE 1

- Unit 1 : Introduction to Data Structure
  - ❖ Algorithms
  - ❖ Complexity
  - ❖ Growth of Functions
- Unit 2 : Array
  - ❖ Array Representation
  - ❖ Basic Operations on Array
  - ❖ Multidimensional Array

Introduction to Data Structures

- A data *type* is a well-defined collection of data with a well-defined set of operations on it. A data *structure* is an actual implementation of a particular abstract data type.
- Abstract Data type : Abstraction can be thought of as a mechanism for suppressing irrelevant details while at the same time emphasizing relevant ones.
- An important benefit of abstraction is that it makes it easier for the programmer to think about the problem to be solved.
- Data abstraction lets the software designer think about the objects in a program and the interactions between those objects without having to worry about how those objects are implemented. Example: Int

Objective:

At the end of the Module, students should be able to,

1. Appreciate data Structures
2. Know the Types of Data Structures
3. Understand the data structures used in C

Data Structure

- Data Structures organizes data helps to create more efficient program
- Any program for a collection of records can be searched, processed in any order or modified
- The choice of data structure and algorithm can make the difference between a program running in few seconds or many days
- A data structure requires,
  - Space for each data item it stores
  - Time to perform each basic operation
  - Programming Effort

Selecting a data structure

Select a data Structure based on the following criteria,

- Analyze the problem to determine the resource constraints a solution must meet
- Determine the basic operations that must be supported. Quantify the resource constraints for each operation
- Select the data structure that best meets these requirements

Some questions to ask?

- Are all data inserted into the data structure at the beginning or are insertions interspersed with other operations?
- Can data be deleted?
- Are all data processed in some well-defined order, or in random access allowed?

Algorithms

- What is an algorithm?

- An algorithm is a finite set of precise instructions for performing a computation or for solving a problem.

- This is a rather vague definition. You will get to know a more precise and mathematically useful definition when you attend CS420.

- But this one is good enough for now…

Algorithms

- Properties of algorithms:

- **Input** from a specified set,
- **Output** from a specified set (solution),
- **Definiteness** of every step in the computation,
- **Correctness** of output for every possible input,
- **Finiteness** of the number of calculation steps,
- **Effectiveness** of each calculation step and
- **Generality** for a class of problems.

Algorithm Examples

- We will use a pseudocode to specify algorithms, which slightly reminds us of Basic and Pascal.
- **Example:** an algorithm that finds the maximum element in a finite sequence

```
•procedure max($a_1$, $a_2$, …, $a_n$: integers)
•max := $a_1$
•for i := 2 to n
•        if max < $a_i$ then max := $a_i$
•{max is the largest element}
```

Algorithm Examples

- Another example: a linear search algorithm, that is, an algorithm that linearly searches a sequence for a particular element.

```
**procedure** linear_search(x: integer; $a_1, a_2, \ldots, a_n$:    integers)
i := 1
**while** ($i \leq n$ and $x \neq a_i$)
        i := i + 1
**if** $i \leq n$ **then** location := i
**else** location := 0
```

- {location is the subscript of the term that equals x, or is zero if x is not found}

Algorithm Examples

•If the terms in a sequence are ordered, a binary search algorithm is more efficient than linear search.

•The binary search algorithm iteratively restricts the relevant search interval until it closes in on the position of the element to be located.

Algorithm Examples

```
procedure binary_search(x: integer; $a_1, a_2, \ldots, a_n$: integers)
i := 1   {i is left endpoint of search interval}
j := n  {j is right endpoint of search interval}
while (i < j)
begin
        m := $\lfloor (i + j)/2 \rfloor$
        if x > $a_m$ then i := m + 1
        else j := m
end
if x = $a_i$ then location := i
else location := 0
•{location is the subscript of the term that equals x, or is zero if x is not found}
```

Complexity

• In general, we are not so much interested in the time and space complexity for small inputs.

• For example, while the difference in time complexity between linear and binary search is meaningless for a sequence with $n = 10$, it is gigantic for $n = 2^{30}$.

Complexity

- For example, let us assume two algorithms A and B that solve the same class of problems.
- The time complexity of A is 5,000n, the one for B is $\lceil 1.1^n \rceil$ for an input with n elements.
- For n = 10, A requires 50,000 steps, but B only 3, so B seems to be superior to A.
- For n = 1000, however, A requires 5,000,000 steps, while B requires $2.5 \cdot 10^{41}$ steps.

Complexity

•This means that algorithm B cannot be used for large inputs, while algorithm A is still feasible.

•So what is important is the **growth** of the complexity functions.

•The growth of time and space complexity with increasing input size n is a suitable measure for the comparison of algorithms.

The Growth of Functions

- The growth of functions is usually described using the **big-O notation**.

- **Definition:** Let f and g be functions from the integers or the real numbers to the real numbers.
- We say that f(x) is O(g(x)) if there are constants C and k such that

- $|f(x)| \leq C|g(x)|$

- whenever $x > k$.

The Growth of Functions

- When we analyze the growth of **complexity functions**, f(x) and g(x) are always positive.

- Therefore, we can simplify the big-O requirement to

- $f(x) \leq C \cdot g(x)$ whenever $x > k$.

- If we want to show that f(x) is O(g(x)), we only need to find **one** pair (C, k) (which is never unique).

The Growth of Functions

•The idea behind the big-O notation is to establish an **upper boundary** for the growth of a function f(x) for large x.

•This boundary is specified by a function g(x) that is usually much **simpler** than f(x).

•We accept the constant C in the requirement

•$f(x) \leq C \cdot g(x)$ whenever $x > k$,

•because **C does not grow with x.**

•We are only interested in large x, so it is OK if $f(x) > C \cdot g(x)$ for $x \leq k$.

The Growth of Functions

Example:

Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

For $x > 1$ we have:

$x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2$

$\Rightarrow x^2 + 2x + 1 \leq 4x^2$

Therefore, for $C = 4$ and $k = 1$:

$f(x) \leq Cx^2$ whenever $x > k$.

$\Rightarrow f(x)$ is $O(x^2)$.

The Growth of Functions

•Question: If f(x) is O(x^2), is it also O(x^3)?

•**Yes.** x^3 grows faster than x^2, so x^3 grows also faster than f(x).

•Therefore, we always have to find the **smallest** simple function g(x) for which f(x) is O(g(x)).

The Growth of Functions

- “Popular” functions g(n) are
- $n \log n$, 1, 2^n, n^2, $n!$, n, n^3, $\log n$

- Listed from slowest to fastest growth:

- 1
- $\log n$
- n
- $n \log n$
- n^2
- n^3
- 2^n
- $n!$

The Growth of Functions

•A problem that can be solved with polynomial worst-case complexity is called **tractable**.

•Problems of higher complexity are called **intractable.**

•Problems that no algorithm can solve are called **unsolvable**.

Arrays : An Overview

Introduction:

Array is one of the simplest data structure in computer programming. Arrays hold a fixed number of equally sized data elements, generally of the same data type.

Objective:

At the end of the Module, students should be able to,

Introduction to Arrays

Multi-dimensional Arrays

Advantages

Limitations

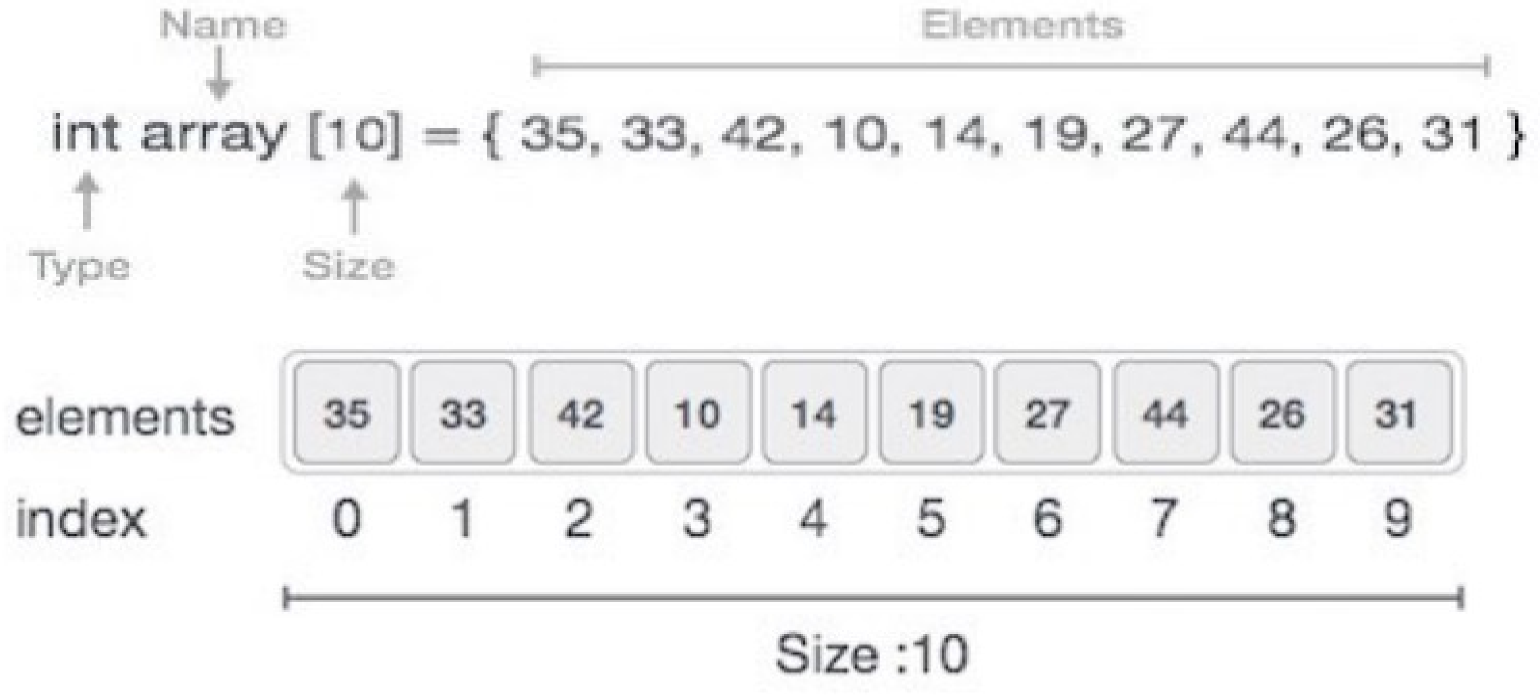
Introduction to Arrays

- Arrays are a data type that are used to represent a large number of homogeneous values, that is values that are all of the one data type.
- The data type could be of type char, in which case we have a string.
- The data type could just as easily be of type int, float or even another array.
- Example : int myarray[] = {1,23,17,4,-5,100};

  The elements in the Array are always stored in consecutive memory locations.
- When data is passed to a function, it is passed by value. But in the case of the array , the actual array is passed to the function and the function can modify it any way it wishes to. The result of the modifications will be available back in the calling program. This is called pass by reference.

Array Representation

- Arrays can be declared in various ways in different languages. For illustration, let's take C array declaration.



- Index starts with 0. Array length is 8 which means it can store 8 elements. Each element can be accessed via its index.

Basic Operations

- Following are the basic operations supported by an array.
- Insertion − add an element at given index.
- Deletion − delete an element at given index.
- Search − search an element using given index or by value.

Insertion Operation

- Insert operation is to insert one or more data elements into an array. Based on the requirement, new element can be added at the beginning, end or any given index of array.
- Let LA is a Linear Array unordered with N elements and K is a positive integer such that K<=N. Below is the algorithm where ITEM is inserted into the K th position of LA
- Algorithm

```
1. Start
2. Set J=N
3. Set N = N+1
4. Repeat steps 5 and 6 while J >= K
5. Set LA[J+1] = LA[J]
6. Set J = J-1
7. Set LA[K] = ITEM
8. Stop
```

Deletion Operation

- Deletion refers to removing an existing element from the array and re-organizing all elements of an array.
- **Algorithm** Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Below is the algorithm to delete an element available at the K th position of LA.

1. Start
2. Set J=K
3. Repeat steps 4 and 5 while J < N
4. Set LA[J-1] = LA[J]
5. Set J = J+1
6. Set N = N-1
7. Stop

Search Operation

- You can perform a search for array element based on its value or its index.
- **Algorithm** Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Below is the algorithm to find an element with a value of ITEM using sequential search.

```
1. Start
2. Set J=0
3. Repeat steps 4 and 5 while J < N
4. IF LA[J] is equal ITEM THEN GOTO STEP 6
5. Set J = J +1
6. PRINT J, ITEM
7. Stop
```

Multi Dimensional Arrays

- The elements of an array can themselves be arrays.
- The following example declares and creates a rectangular integer array with 10 rows and 20 columns

```
int a[][] = new int[10][20];
for (int i = 0; i < a.length; i++)
{
      for (int j = 0; j < a[i].length; j++)
       {
                a[i][j] = 0;
       }
}
```

- The elements are accessed as a[i][j]. This is a consistent notation since a[i][j] is element j of the array a[i].

Advantages & Limitations

Advantages

- Arrays are,
  - Simple and easy to understand
  - Contiguous allocation
  - Fast retrieval because of its indexed nature
  - No need for the user to be worried about the allocation and de-allocation of arrays

Limitations

- If you need m elements out of n locations defined,
  - n-m locations are unnecessarily wasted if n>m

    or
  - an error occurs if m>n named out of bounds error.

Exercise

1. Define data structures. Give some examples.
2. In how many ways can you categorize data
3. structures? Explain each of them.
4. Discuss the applications of data structures.
5. What is the use of multi-dimensional arrays?
6. Explain sparse matrix.
7. How is an array represented in the memory?
8. How is a two-dimensional array represented in the memory?
9. For an array declared as int arr[50], calculate the address of arr[35], if Base(arr) = 1000 and w = 2.
10. Write a algorithm to find the median of n numbers. Find number of instruction executed by your algorithm. What are the time and space complexities?
11. Write an algorithm to sort elements by bubble sort algorithm. What are the time and space complexities?

References

- Patel, Mayank. *Data Structure and Algorithm With C*. Educreation Publishing, 2018.
- E.Horowitz and S.Sahani, “Fundamentals of Data structures”, Galgotia Book source Pvt. Ltd., 2003.
- R.S.Salaria, “Data Structures & Algorithms”, Khanna Book Publishing Co. (P) Ltd..,2002.
- Y.Langsam et. Al., “Data Structures using C and C++”, PHI, 1999.
- Bergin, Joseph A. *Data Abstraction: The Object-Oriented Approach Using C++/Book and Disk*. McGraw-Hill, Inc., 1994.
- Samet, Hanan. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

Declaration

"The content is exclusively meant for academic purpose and for enhancing teaching and learning. Any other use for economic/commercial purpose is strictly prohibited. The users of the content shall not distribute, disseminate or share it with anyone else and its use is restricted to advancement of individual knowledge. The information provided in this e-content is authentic and best as per knowledge".

Dr. Dharm Raj Singh
Assistant Professor, (HOD)
Department of Computer Application
JagatpurP. G. College, Varanasi

Thanks