Data Structures and Algorithms

W1- Lecture 1, 2

Introduction

Engr. Bushra Tahir
Department of Electrical Engineering
Iqra National University

Text and Reference Books

1. Horowitz Sahni, "Fundamentals of Data Structures in C++", 1999.
2. 2. Lipshutz, "Data Structures", Schaum Outline Series, 1999.
3. 3. Weiss, "Data structures and algorithm analysis in C++".
4. 4. A. M. Tanenbaum, "Data structures using C and C++", 2001.

Grading Policy

- Assignments (10 Marks)
- Quiz (10 Marks)
- Mid Term Exam (30 Marks)
- Final Term Exam (50 Marks)

Goals

- In this course, we will do problem solving and see that the organization of data in some cases is of immense importance.
- The data will be stored in a special way so that the required result should be calculated as fast as possible.
- Cover well-known data structures such as dynamic arrays, linked lists, stacks, queues, trees and graphs.
- Implement data structures in C++

Definition of Data Structures

- Data structures help us to organize the data in the computer, resulting in more efficient programs. An efficient program executes faster and helps minimize the usage of resources like memory, disk

Use of Data Structures

- Computers are getting more powerful.
- People have started solving more complex problems.
- Computer applications are becoming complex, so more resources are requires.
- Instead of buying new computers, use programming, data structures and algorithms for applications to execute faster.

What does organizing the data mean?

- Data should be arranged in a way that it is easily accessible.
- Suppose the data contains some numbers and the programmer wants to calculate the average, standard deviation etc.
- To solve such problems, data structures and algorithm are used.

What does organizing the data mean?

Data are also organized into more complex types of structures. The study of such data structure, which forms the subject matter of the text, includes the following three steps:

1. Logical or mathematical description of the structure.
2. Implementation of the structure on a computer.
3. Quantitative analysis of the structure, which include determining the amount of memory needed to store the structure and the time required to process the structure.

Solution Efficiency and Resources

- A solution is said to be efficient if it solves the problem within its resource constraints.
- Resources include;
  - ❑ More Memory
  - ❑ Faster CPU
  - ❑ Less time

Selection of Data Structure

- Analyze the problem to determine the resource constraints that 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

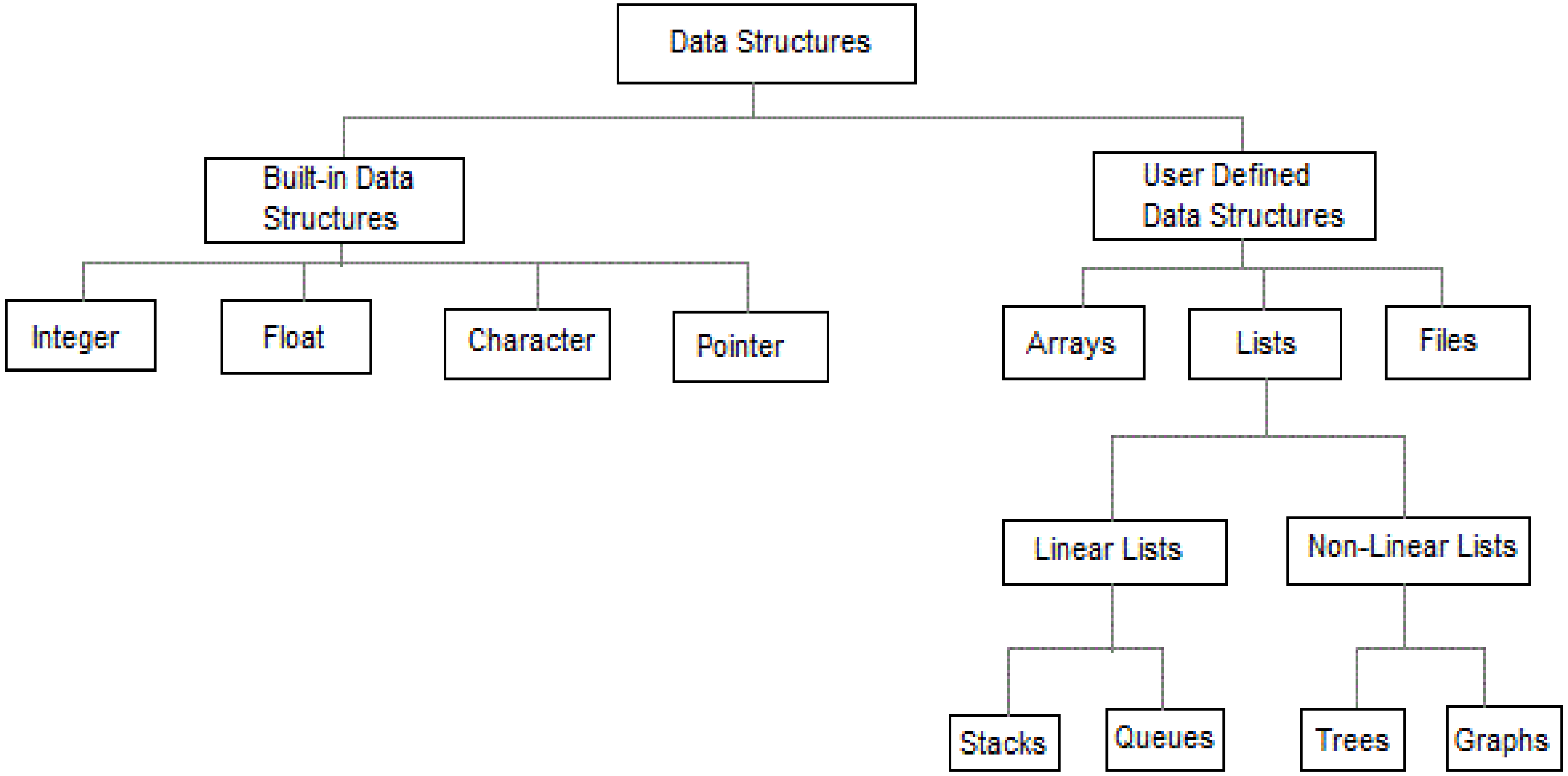
Selection of Data Structure

- Suppose there are eight million names in the directory. Now someone asks you about the name of some particular person. You want that this query should be answered as soon as possible. You may add or delete some data. It will be advisable to consider all these operations when you select some data structure.
- The data is so huge i.e. in Giga bytes (GBs) while the disc space available with us is just 200 Mega bytes. This problem can not be solved with programming. Rather, we will have to buy a new disk.

Philosophy of Data Structure

- Each data structure has costs and benefits.
- Any data structure used in your program will have some benefits.
- There are three basic things associated with data structures. A data structure requires:
  - space for each data item it stores
  - time to perform each basic operation
  - programming effort

Data Types


Data Structures
Built-in Data Structures
User Defined Data Structures
Integer
Float
Character
Pointer
Arrays
Lists
Files
Linear Lists
Non-Linear Lists
Stacks
Queues
Trees
Graphs

Data Types

- Built in data structures are called Primitive data structures.
- User defined data structures are called Abstract data structures.
- In Linear data structures, the data items are arranged in a linear sequence
- In Non-Linear data structures, the data items are not in sequence.

Data Structure Operations and Algorithms Complexity

Data Structure Operations

- ***Search:*** By means of this operation, we can find the location of the element with a given value or the record with a given key.
- ***Insertion:*** Insertion operation is used for adding a new element to the list.
- ***Deletion:*** Deletion operation is used to remove an element from the list.
- ***Sorting:*** This operation is used for arranging the elements in some type of order.
- ***Merging:*** By means of merging operation, we can combine two lists into a single list.

Algorithm Complexity

- An algorithm is a clearly specified set of simple instructions to be followed to solve a problem.
- An algorithm is a procedure that you can write as a C function or program, or any other language.
- Once an algorithm is given for a problem and decided (somehow) to be correct, an
- important step is to determine how much in the way of resources, such as time or space, the
- algorithm will require.

Algorithm Complexity

- An algorithm that solves a problem but requires a year is hardly of any use.
- An algorithm states explicitly how the data will be manipulated. Some algorithms are more efficient than others

Algorithm Complexity

- The complexity of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data the algorithm must process.
- There are two main complexity measures of the efficiency of an algorithm
  - ❑ Time Complexity
  - ❑ Space Complexity

Time Complexity

- Time complexity is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm.
- “Time” can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take.
- The better the time complexity of an algorithm is, the faster the algorithm will carry out his work in practice

Space Complexity

- Space complexity is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm.
- This is essentially the number of memory cells which an algorithm needs.
- A good algorithm keeps this number as small as possible, too.

Big O Notation

- “Big O” refers to a way of rating the efficiency of an algorithm.
- It is only a rough estimate of the actual running time of the algorithm, but it will give you an idea of the performance relative to the size of the input data.
- The time efficiency of almost all of the algorithms can be characterized by only a few growth rate functions.

O(l) - constant time

This means that the algorithm requires the same fixed number of steps regardless of the size of the task.

- ❑Push and Pop operations for a stack (containing n elements);
- ❑Insert and Remove operations for a queue.

O(l) - constant time

- ❑ *Sequence of statements*

  statement 1;

  statement 2;

  ... statement k;

  The total time is found by adding the times for all statements: total time = time(statement 1) + time(statement 2) + ... + time(statement k)

  If each statement is "simple" (only involves basic operations) then the time for each statement i

O(n) - linear time

This means that the algorithm requires a number of steps proportional to the size of the task.

- Finding the maximum or minimum element in a list
- Sequential search in an unsorted list of n elements
- Calculating iteratively n-factorial

O(n) - linear time

```
if
block 1 (sequence of statements)
else
block 2 (sequence of statements)
end if;
```

Here, either block 1 will execute, or block 2 will execute. Therefore, the worst-case time is the slower of the two possibilities:

max(time(block 1), time(block 2))

If block 1 takes O(1) and block 2 takes O(N), the if-then-else statement would be O(N)

O(n^2) - quadratic time

- The number of operations is proportional to the size of the task squared.
- Nested loops

```
for I in 1 .. N loop
for J in 1 .. M loop
sequence of statements
end loop;
end loop;
```

O(n^2) - quadratic time

- The outer loop executes N times. Every time the outer loop executes, the inner loop executes M times. As a result, the statements in the inner loop execute a total of N * M times. Thus, the complexity is O(N * M).
- If inner loop also executes N times, the total complexity for the two loops id O(N^2).

O(log n) - logarithmic time

- An input data set containing 10 items takes one second to complete
- A data set containing 100 items takes two seconds, and a data set containing 1000 items will take three seconds.
- Doubling the size of the input data set has little effect on its growth
- This makes algorithms like binary search extremely efficient when dealing with large data sets.

O(log n) - logarithmic time

- Binary search is a technique used to search sorted data sets.
- It works by selecting the middle element of the data set and compares it against a target value.
- If the values match it will return success.
- If the target value is higher than the value of the probe element it will take the upper half of the data set and perform the same operation against it.

O(log n) - logarithmic time

- if the target value is lower than the value of the probe element it will perform the operation against the lower half.
- It will continue to halve the data set with each iteration until the value has been found or until it can no longer split the data set.

Questions?