Data Structures

Lecture 2 :

Part 1: Programming Methodologies

Part 2: Memory Management

Dr. Essam Halim Houssein
Lecturer, Faculty of Computers and Informatics,
Benha University

Part 1: Programming Methodologies

Programming methodologies deal with different methods of designing programs. This will teach you how to program efficiently.

Data is the basic entity or fact that is used in calculation or manipulation process. There are two types of data such as numerical and alphanumerical data. Integer and floating-point numbers are of numerical data type and strings are of alphanumeric data type. Data may be single or a set of values, and it is to be organized in a particular fashion.

This organization or structuring of data will have profound impact on the efficiency of the program.

Part 1: Programming Methodologies

1.1. AN INTRODUCTION TO DATA STRUCTURE

Data structure is the structural representation of logical relationships between elements of data. In other words a data structure is a way of organizing data items by considering its relationship to each other.

Algorithm + Data Structure = Program

Data Structure = Organized data + Operations

Part 1: Programming Methodologies

1.2 Algorithm

Algorithm is a step-by-step finite sequence of instruction, to solve a well-defined computational problem.

That is, in practice to solve any complex real life problems; first we have to define the problem. Second step is to design the algorithm to solve that problem.

Part 1: Programming Methodologies

1.3. STEPWISE REFINEMENT TECHNIQUES

There are three steps in refinement process, which is illustrated in Fig. 1.1.

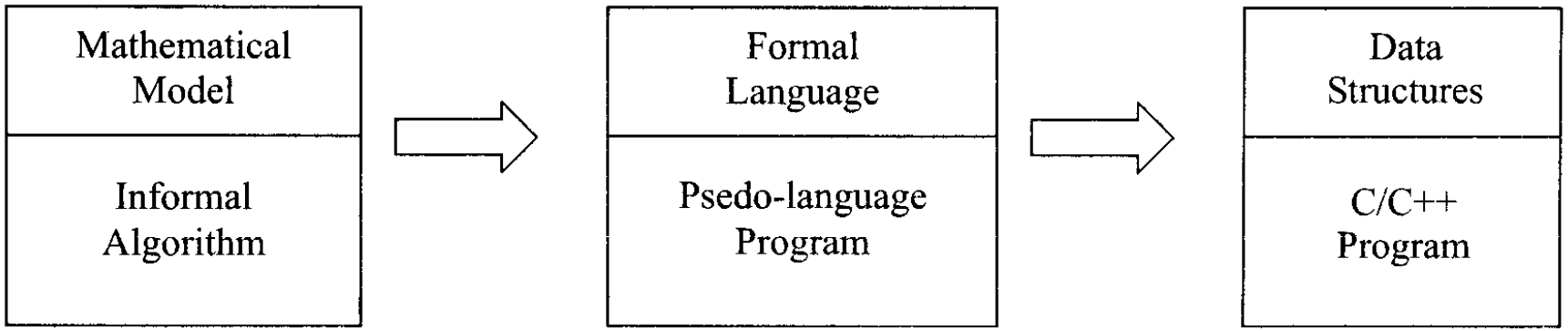

Fig. 1.1

Part 1: Programming Methodologies

1.4. MODULAR PROGRAMMING

Modular Programming is heavily procedural. The focus is entirely on writing code (functions). Any code may access the contents of any data structure passed to it. Modular Programming is the act of designing and writing programs as functions, that each one performs a single well-defined function, and which have minimal interaction between them. That is, the content of each function is cohesive, and there is low coupling between functions.

Part 1: Programming Methodologies

Two methods may be used for modular programming. They are known as top-down and bottom-up, regardless of whether the top-down or bottom-up method is used, the end result is a modular program. This end result is important, because not all errors may be detected at the time of the initial testing.

Errors, bugs, debugging ???

Part 1: Programming Methodologies

1.5. TOP-DOWN ALGORITHM DESIGN

The principles of top-down design dictates that a program should be divided into a main module and its related modules. Each module should also be divided into sub modules according to software engineering and programming style. The division of modules processes until the module consists only of elementary process that are intrinsically understood and cannot be further subdivided.

Part 1: Programming Methodologies

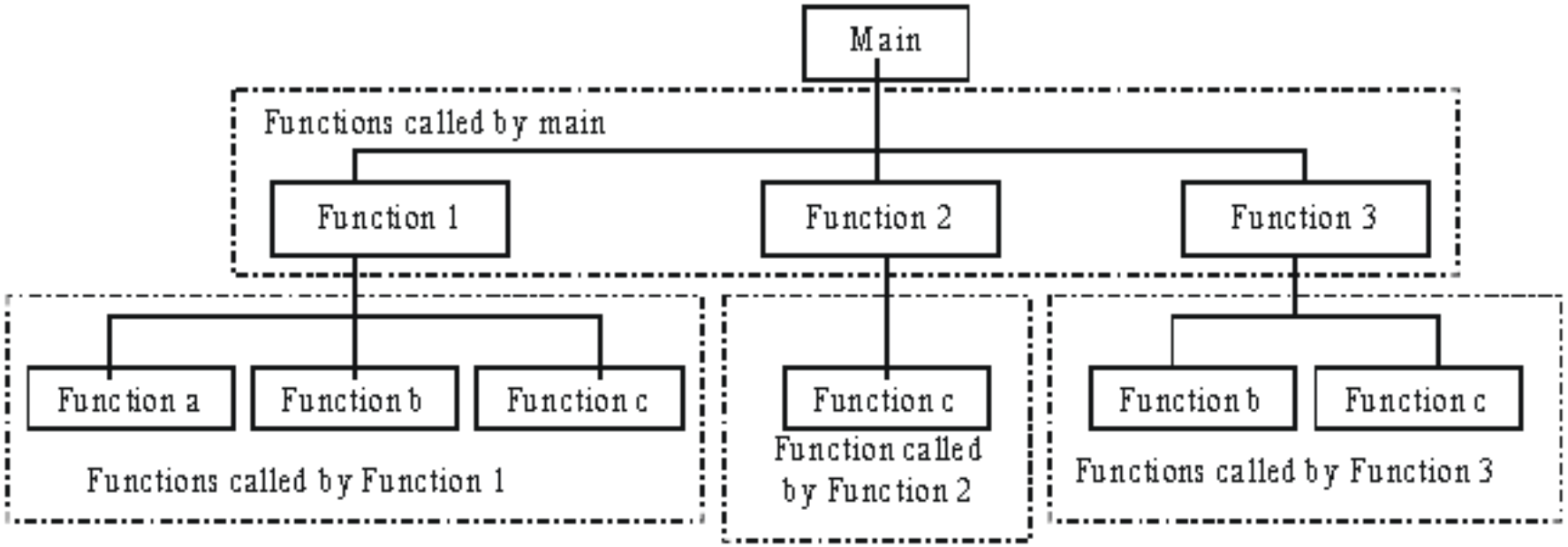

Fig. 1.2

Part 1: Programming Methodologies

1.6. BOTTOM-UP ALGORITHM DESIGN

Bottom-up algorithm design is the opposite of *top-down design*. It refers to a style of programming where an application is constructed starting with existing primitives of the programming language, and constructing gradually more and more complicated features, until the all of the application has been written. That is, starting the design with specific modules and build them into more complex structures, ending at the top.

Part 1: Programming Methodologies

The bottom-up method is widely used for testing, because each of the lowest-level functions is written and tested first. This testing is done by special test functions that call the low-level functions, providing them with different parameters and examining the results for correctness. Once lowest-level functions have been tested and verified to be correct, the next level of functions may be tested. Since the lowest-level functions already have been tested, any detected errors are probably due to the higher-level functions. This process continues, moving up the levels, until finally the main function is tested.

Part 1: Programming Methodologies

1.7. STRUCTURED PROGRAMMING

It is a programming style; and this style of programming is known by several names: Procedural decomposition, Structured programming, etc.

Structured programming is not programming with structures.

Part 1: Programming Methodologies

1.8. ANALYSIS OF ALGORITHM

After designing an algorithm, it has to be checked and its correctness needs to be predicted; this is done by analyzing the algorithm. The algorithm can be analyzed by tracing all step-by-step instructions. That is, design the algorithm in a simple way so that it becomes easier to be implemented. Moreover there may be more than one algorithm to solve a problem. The choice of a particular algorithm depends on following performance analysis and measurements :

1. Space complexity

2. Time complexity

Part 1: Programming Methodologies

When we analyze an algorithm it depends on the input data, there are three cases :

1. Best case **2. Average case** **3. Worst case**

In the best case, the amount of time a program might be expected to take on best possible input data.

In the average case, the amount of time a program might be expected to take on typical (or average) input data.

In the worst case, the amount of time a program would take on the worst possible input configuration.

Part 1: Programming Methodologies

1.10. BIG "OH" NOTATION

Big Oh is a characteristic scheme that measures properties of algorithm complexity performance and/or memory requirements. The algorithm complexity can be determined by eliminating constant factors in the analysis of the algorithm. Clearly, the complexity function f(n) of an algorithm increases as 'n' increases.

Part 1: Programming Methodologies

1.12. CLASSIFICATION OF DATA STRUCTURE

Data structures are broadly divided into two :

1. Primitive data structures : These are the basic data structures and are directly operated upon by the machine instructions, which is in a primitive level. They are integers, floating point numbers, characters, string constants, pointers etc.

Part 1: Programming Methodologies

1.12. CLASSIFICATION OF DATA STRUCTURE

2. Non-primitive data structures : It is a more sophisticated data structure emphasizing on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items. Array, list, files, linked list, trees and graphs fall in this category.

Part 1: Programming Methodologies

1.12. CLASSIFICATION OF DATA STRUCTURE

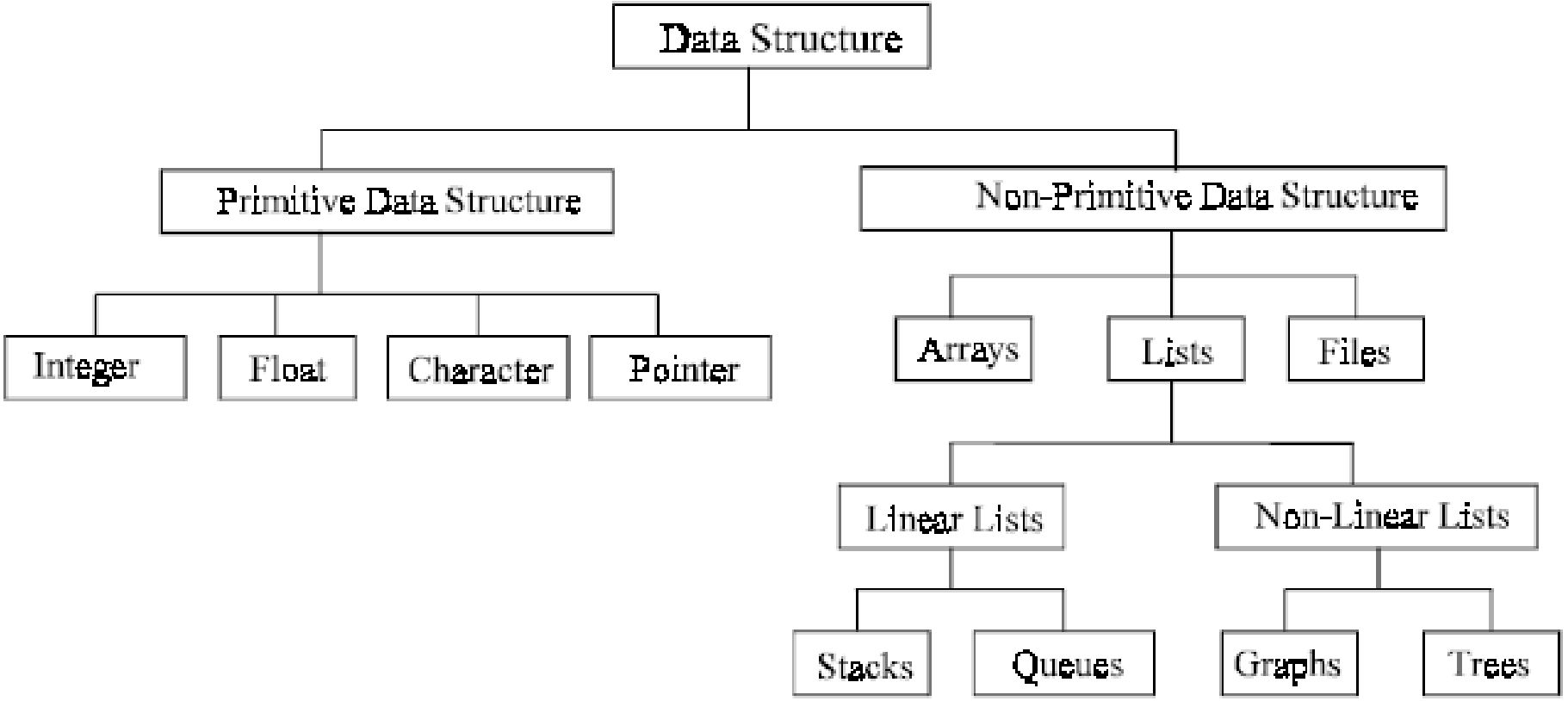

Fig. 1.4. Classifications of data structures

Part 1: Programming Methodologies

1.15. LISTS

an array is an ordered set, which consist of a fixed number of elements.

No deletion or insertion operations. Another main disadvantage is its

fixed length; we cannot add elements to the array.

Part 1: Programming Methodologies

1.15. LISTS

A list is an ordered set consisting of a varying number of elements to which insertion and deletion can be made. List can be implemented by using pointers. Each element is referred to as nodes; therefore a list can be defined as a collection of nodes as shown below :

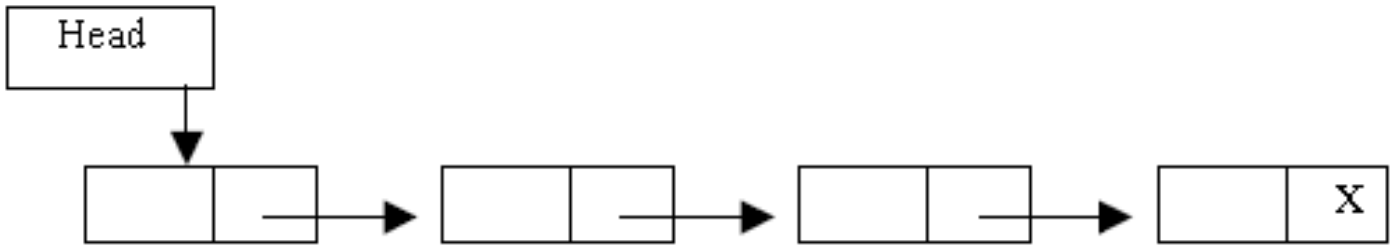

Fig. 1.7

Part 1: Programming Methodologies

1.16. FILES AND RECORDS

A file is typically a large list that is stored in the external memory (e.g., a magnetic disk) of a computer.

A record is a collection of information (or data items) about a particular entity. More specifically, a record is a collection of related data items, each of which is called a filed or attribute and a file is a collection of similar records.

Although a record is a collection of data items, it differs from a linear array in the following ways:

A. A record may be a collection of non-homogeneous data; i.e., the data items in a record may have different data types.

B. The data items in a record are indexed by attribute names, so there may not be a natural ordering of its elements.

Part 2: Memory Management

A memory or store is required in a computer to store programs (or information or data). Data used by the variables in a program is also loaded into memory for fast access. A memory is made up of a large number of cells, where each cell is capable of storing one bit. The cells may be organized as a set of addressable words, each word storing a sequence of bits. These addressable memory cells should be managed effectively to increase its utilization.

Part 2: Memory Management

That is memory management is to handle request for storage and release of storage in most effective manner. While designing a program the programmer should concentrate on to allocate memory when it is required and to deallocate once its use is over. In other words, dynamic data structure provides flexibility in adding, deleting or rearranging data item at run-time. Dynamic memory management techniques permit us to allocate additional memory space or to release unwanted space at run-time, thus optimizing the use of storage space.

Part 2: Memory Management

2.2. DYNAMIC MEMORY ALLOCATION IN C++

Although C++ supports all the functions (i.e., malloc, calloc, realloc and free) used in C, it also defines two unary operators **new and delete** that performs the task of allocating and freeing the memory in a better and easier way. **An object** (or variable) can be created by using **new**, and destroyed by using **delete**, as and when required. **A data object created inside a block with new, will remain in existence until it is explicitly destroyed by using delete.**

Part 2: Memory Management

2.3. FREE STORAGE LIST

To store any data, memory space is allocated dynamically. That is storage allocation is done when the programmer requests it by declaring a structure at the run time.

But freeing storage is not as easy as allocation. When a program or block of program (or function or module) ends, the storage allocated at the beginning of the program will be freed. Dynamically a memory cell can be freed using the operator delete in C++.

Part 2: Memory Management

Two problems arise in the context of storage release.

One is the **accumulation** of garbage (called garbage collection)

and another is that of **dangling reference**,

Part 2: Memory Management

2.4. GARBAGE COLLECTION

Suppose some memory space becomes reusable when a node (or a variable) is deleted from a list or an entire list is deleted from a program. Obviously, we would like the space to be made available for future use. One way to bring this about is to immediately reinsert the space into the free-storage list-using delete or free. However, this method may be too time-consuming for the operating system and most of the programming languages, reserve themselves the task of storage release, even if they provide operator like delete. So the problem arises when the system considers a memory cell as free storage.

Part 2: Memory Management

2.5. DANGLING REFERENCE

A dangling reference is a pointer existing in a program, which still accesses a block of memory that has been freed. For example consider the following code in C++.

```
int ptr,temp;
------------------
------------------
ptr = new int;
------------------
------------------
temp = ptr
-------------------
-------------------
delete ptr;
-------------------
-------------------
```

Part 2: Memory Management

Here **temp** is the dangling reference. temp is a pointer which is pointing to a memory block ptr, which is just deleted. This can be overcome by using a **reference counters**.

Part 2: Memory Management

2.6. REFERENCE COUNTERS

In the reference-counter method, a counter is kept that records how many pointers have direct access to each memory block. When a memory block is first allocated, its reference counter is set to 1. Each time another link is made pointing to this block, the reference counter is incremented. Each time a link to its block is broken, the reference counter is decremented. When the count reaches 0, the memory block is not accessed by any other pointer and it can be returned to the free list.

This technique completely eliminates the dangling reference problem.

Part 2: Memory Management

2.7. STORAGE COMPACTION

Storage compaction is another technique for reclaiming free storage. Compaction works by actually moving blocks of data from one location in the memory to another so as to collect all the free blocks into one single large block. Once this single block gets too small again, the compaction mechanism is called gain to reclaim the unused storage.

Part 2: Memory Management

2.8. BOUNDARY TAG METHOD

Boundary tags are data structures on the boundary between blocks in the heap from which memory is allocated. The use of such tags allow blocks of arbitrary size to be used as shown in the Fig. 2.1.

Part 2: Memory Management

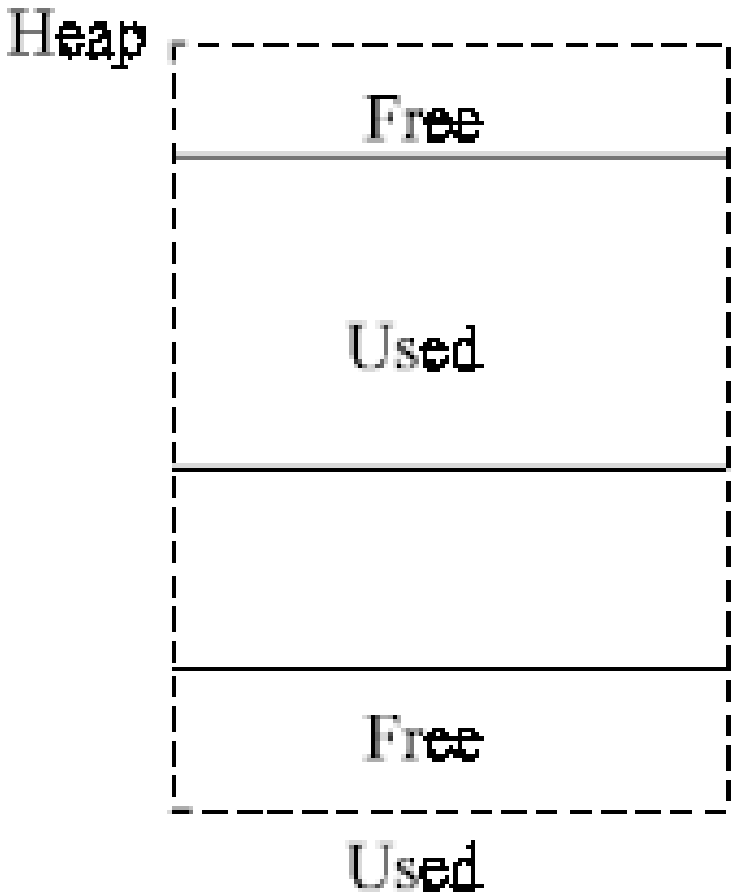

Fig. 2.1

Any Questions?