Lecture 12 Review

**Variable specifiers, memory, scope**

* Variables can be specified by key words:
  + Auto, static, register, extern
  + Auto
    - For local variable, omit
    - Ex. auto char c, char c
  + Static
    - Declare static variable in function
    - Ex. Static int a
  + Register
    - Store variable in a register
    - Ex. Register int b;
  + Extern
    - A global variable in another source file
    - Ex. Extern int d;

**Memory allocations**

* Use variable to represent data and its memory block. Three methods to do memory allocations
  + Automatic
    - Declare function local variable
    - Stack region
    - Released after function call
  + Static
    - Declare global, or static variable
    - Data region
    - Not released
  + Dynamic
    - Point = malloc()
    - Heap region
    - Released by call free(pointer)

**Data structure principles**

* Data structures define how data are:
  + Represented
  + Organized
  + Stored
  + Operated
* Data structures are used in algorithms and programs to represent data values of certain types together with operations that can be performed on the data

**Terms on data**

1. A *data item* (*or data object*) means a data value. A data item is called an *elementary item* if it does not have subordinate data items, otherwise it is called a *composite item*.
2. A *record* is formed by a group of data items
3. Application data is usually a *collection* of records

**Terms on data structures**

1. Abstract data type (ADT): a mathematical model for a certain type of data, whose behaviour is defined by a set of values and a set of operations. Ex. 32-bit INTEGER-ADT, IEEE 754 is an ADT for single procession floating point numbers.
2. Data type: an implementation of an ADT in a programming language. Ex. Int type is an implementation of 32-bit INTEGER-ADT in c. Float type is an implementation of IEEE 754 in c.
3. Abstract data structure: a specification on representation, organization, and operations of a collection of data items. Ex. ARRAY, STACK, QUEUE
4. Data structure: an implementation of an abstract data structure in certain program language. Ex. Int array [n] is a data structure for n int type data values

**Data type**

1. The implementation of an ADT contains detailed bit pattern representation of data values and operations on the bit patterns.
2. Some data type operations are implemented by hardware circuitry as part of the computer CPU, ex. The int addition.
3. Some other operations are implemented by software in which programs are used to interpret bit patterns and to perform the operations.

**Data type categories**

1. Native data types: data types directly supported by a computer CPU for manipulation bit pattern as binary numbers.
2. Primitive data types: data types directly supported by a program language (built-in data types). A primitive data type defines how the data will be internally represented, stored, accessed and operated. Primitive data types are the basic data types to build other data types.
3. Structured data types: A structure data type is a composite data type consisting of one or more data elements (member fields) of various types.
   1. A structured data type generally has two parts : data part and address part. The data part contains data values of various data types. The address part stores runtime location information of other data objects.
   2. An atomic type is a structure data type that contains no address part, namely only the data items.

**Data structure operations**

* Basic operations
  + Accessing: to get value of any data element in a data structure
  + Modifying: to set value of a data element
* Common operations
  + Traversal: to access each data element exactly once
  + Searching: to find the location of one or more data elements that satisfy the given condition
  + Inserting: to add a new data element to a data structure
  + Deleting: to remove a particular data element from a data structure
* Advanced operations
  + Sorting: to arrange data objects in ascending or descending order of keys
  + Merging: to combine two collections of data values into one data structure

**Application specific data structures**

* When designing an application algorithm, abstract data type/structures are used to represent data
* An application algorithm uses data operations on the abstract data types/structures
* When writing a program to implement an algorithm, it needs to use primitive data types to implement the abstract data types/structures to create the algorithm specific data types/structure so that the operations can be done efficiently. We call such data structures application specific data structures.

**Algorithms and data structures**

* An algorithm is a step-by-step computational procedure to solve a problem by performing calculations on input data and producing results in output. Abstract data types/structures are usually used to represent input data, intermediate data, and output data in algorithms
* A program is an implementation of an algorithm in a programming language. Abstract data types and abstract data structures are translated to data types and data structures to represent and access data in the program.
* Algorithm analysis concerns *correctness, time, and space*

**Space complexity**

* Space complexity of an algorithm is the total space taken by the algorithm with respect to the input size. Space complexity includes both auxiliary space and space used by input
* Auxiliary space is the extra space or temporary space used by an algorithm. When input size is given, we want to design algorithm which uses less auxiliary space, so we usually analyze auxiliary space complexity in terms of space complexity.

**Data structure designs, analysis, and implementations**

1. Designing application specific data structures usually starts from designing abstract data types/structures to meet the needs of data representations of application algorithms.
2. Analysis of algorithms
3. The implementations of the abstract data types/structures are done when writing programs to implement the algorithm.

**Classification of data structures**

* Primitive data structures
  + A primitive data structure consists of a single data element of a primitive data type of a programming language
  + Ex. Char, int, float are some primitive data types in c , any variables of these types are primitive data structures
* Non-primitive data structures
  + A non-primitive data structures is created using primitive data structures.
  + Ex. List, array, linked lists, stacks, etc.
* Non-primitive data structures can be further classified into two categories:
  + Linear: has data elements organized or stored in sequential order
    - Ex. Arrays, linked lists, queues, stacks
  + Nonlinear
    - Ex. Trees, heaps, graphs

**Linear data structures**

* Linear data structures can be represented in memory in two different methods:
  + Physically ordered: two adjacent elements in the ordering are stored adjacently in memory eg. Arrays
  + Virtually linked: the linear relationship between elements is defined by means of links eg. Addresses of next data elements
* Linked data structures are data structures in which relations of elements (nodes) are maintained by linked. Keeping links is a major task for data structures operations

**Limitations of using arrays**

* Arrays are used when the maximum number of elements to store is known
* Arrays are efficient for random access. Operations like traversal, searching and sorting can also be performed efficiently on arrays.
* But arrays have limitations. Arrays are of fixed length, it can cause an overflow if the number of records to store is bigger than the length and wastes memory if number of records is much smaller than the length.
* Insert and delete operations may have high costs because of shifting elements from their positions.

**Midterm Coverage**

* Midterm time: 8:30-9:20 Monday feb 22
* Closed book
* Questions types:
  + 25 selection questions
  + 4 short answer questions
  + 4 short C program questions
* Coverage: lectures 1-12

**Ex. Briefly describe how compiling works**

* Preprocessing
* Compiling
* Assembly
* Linking

