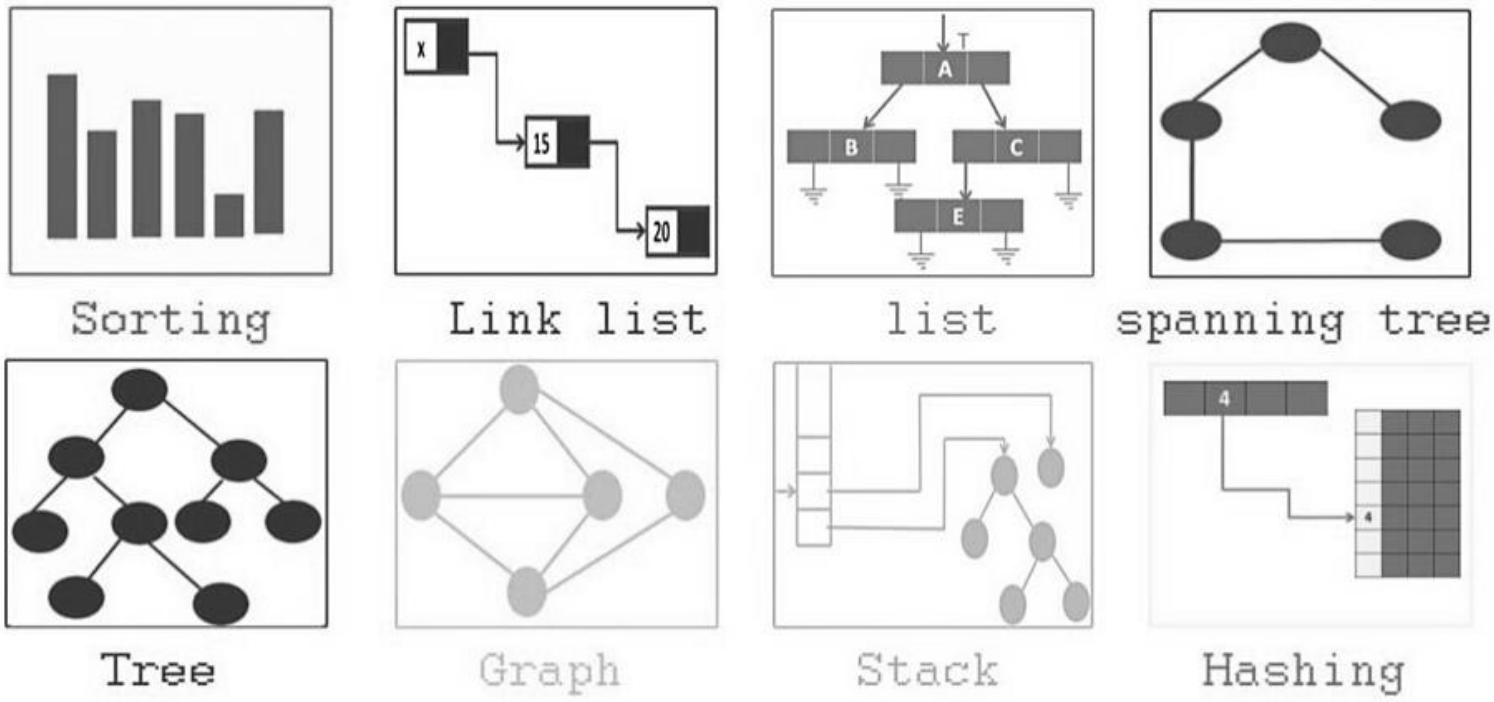


CS 2124: Data Structures
Spring 2024

- First Lecture

Instructor
- Dr. Murtaza Siddiqi

Google Scholar Link

Topics (First week)

- Introduction (Course and Outline)
- Why study Data Structures
- Data Structure
  - Types of data structures
  - Operations on data structures
- Planning to program (Revision)
  - Algorithm, Pseudocode and Flow chart
  - How to solve problems (using programming)
  - Structures of programming
- C Language (Revision)
  - Why use C to teach data Structures
  - Review Data types
  - Arrays, Pointers, Array and Pointers, 2D Arrays, Structures & Union

Students who are new to C-language should try to get familiar with C language during this week

Introduction: About the Course

- **Objective:** The objective is to enable the students to use the discussed techniques in their programs to efficiently retrieve, store and manipulate data and also to enable them to study advanced data structures and related algorithms on their own.

- **Learning Goals:**
  - To provide the knowledge of basic data structures with their implementations and applications.
  - To understand the importance of data structures in context of writing effective programs.
  - To develop skills to apply suitable data structures in problem solving and optimizing programs.

- Recommended Books:
  1. Data Structures Using C and C++, by Langsam, Augenstein, and Tenenbaum
  2. Data Structures and Algorithm Analysis in C, Mark Allen Weiss

Introduction: Grading

Marks Head	Total Frequency	Marks /Frequency	Total Marks /Head
Assignments	4	5	20%
Final Exam (7th May 2024)	1	20	20%
Midterm Exam (29th Feb 2024)	1	20	20%
Lab Work*	N/A*	20	20%
Quizzes	2	5	10%
Attendance/Class participation	2	5	10%

***Lab work** = Lab points will be based on attendance, lab task, quiz etc. The TA will inform the students about the assessment and points criteria*

- **Sessions:** Tuesday and Thursday
- Grades will be as per **UTSA defined scheme** (default grading scheme).
- Course Outline is available on Canvas and UTSA Bluebook (Link)
- Exam and Quiz will be discussed in class
- Lecture slides will be uploaded on Canvas after Thursday's lecture (PDF format)
- Be careful with assessment deadlines
- Retake of Quiz and Exam will be as per UTSA policy

Any Questions

Why Study Data Structures or Importance of Data Structures for a CS student (Practical Application).

- Big Data
- Data is everything and it must be managed to extract information
- Applications, websites must be optimized (Data access)

In real life data is involved in Banking, Communications, Media and Entertainment, Healthcare Providers, Education, Manufacturing, Government, Insurance, Retail, Wholesale trade, Transportation, Energy and Utilities

Why Study Data Structures or Importance of Data Structures for a CS student.

- Fundamental subjects in Computer Science.
- Helps in developing your thinking in terms of code (problem solving).
- Helps in solving the same problem but with better time complexities (As we will study the space and time complexity or Big O notation)

- Most of the self-taught programmers miss the fundamentals of Computer Science. They jump directly to the development process (in-demand skill in the industry).
- The most popular technologies can change overnight, but the fundamentals like Data Structures or Analysis of Algorithms stay.

Data Structures

- **What is Data Structure ?**
- In programming, the term data structure refers to a scheme for organizing related pieces of information.

- The basic types of data structures include:
  - Files / lists
  - Arrays / Records
  - Trees / Tables
  - Graphs etc.

Types of Data Structures

- There are numerous types of data structures, generally built upon simpler primitive data types:

1. An **array** is a number of elements in a specific order, typically all of the same type.
2. A **record** (also called a tuple or struct) is an combined data structure. A record is a value that contains other values, typically in fixed number and sequence and typically indexed by names. The elements of records are usually called fields or members.
3. An **associative array** (also called a dictionary or map) is a more flexible variation on an array, in which name-value pairs can be added and deleted freely.

*(A **name–value pair**, also called an attribute–value pair, key–value pair, or field–value pair, is a fundamental data representation in computing systems and applications)*

Array

```c
#include <stdio.h>
int main() {

  int values[5];

  printf("Enter 5 integers: ");

  // taking input and storing it in an array
  for(int i = 0; i < 4; ++i) {
    scanf("%d", &values[i]);
  }

  printf("Displaying integers: ");

  // printing elements of an array
  for(int i = 0; i < 5; ++i) {
    printf("%d\n", values[i]);
  }
  return 0;
}
```

Associative Array

Associative arrays, also called maps or dictionaries, are an abstract data type that can hold data in (key, value) pairs.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    const char *months[5][2] =
    {
        {"January", "Jan"},
        {"March", "Mar"},
        {"August", "Aug"},
        {"September" , "Sep"},
        {"October", "Oct"},
        {"December", "Dec"}
    };
int x;
for(x=0; x<5; x++)
printf("%s = %s \n", months[x][0], months[x][1]);
}
```

- Will the Program compile?
- Will the Program Run after compilation?
- Will there be an error?
- Will there be a warning?

Associative Array

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    const char *months[5][2] =
    {
        {"January", "Jan"},
        {"March", "Mar"},
        {"August", "Aug"},
        {"September" , "Sep"},
        {"October", "Oct"},
        {"December", "Dec"}
    };
int x;
for(x=0; x<5; x++)
printf("%s = %s \n", months[x][0], months[x][1]);
}
```

	0	1
0	January	Jan
1	March	Mar
2	August	Aug
3	September	Sep
4	October	Oct

Associative Array

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    const char *months[6][2] =
    {
        {"January", "Jan"},
        {"March", "Mar"},
        {"August", "Aug"},
        {"September" , "Sep"},
        {"October", "Oct"},
        {"December", "Dec"}
    };
int x;
for(x=0; x<6; x++)
printf("%s = %s \n", months[x][0], months[x][1]);
}
```

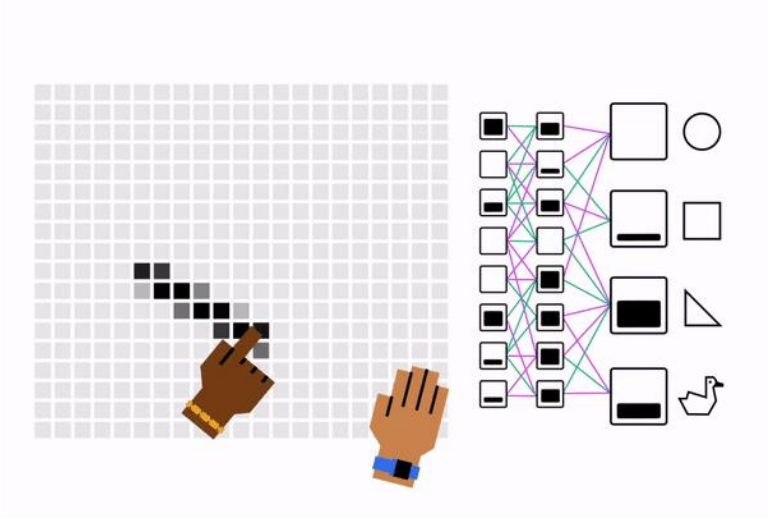
Types of Data Structures

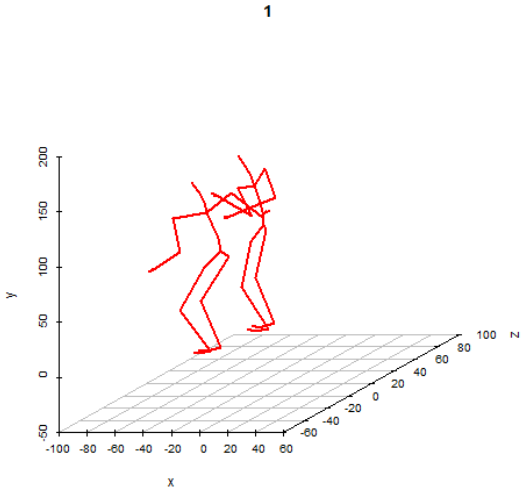
- A ***structure*** contains an ordered group of data objects. Unlike the elements of an array, the data objects within a structure can have varied data types. Each data object in a structure is a *member* or *field*.

- A ***union*** is an object similar to a structure except that all of its members start at the same location in memory. A union variable can represent the value of only one of its members at a time.

Types of Data Structures

- **Graphs and Trees** are linked abstract data structures composed of nodes.
- Each node contains a value and one or more pointers to other nodes arranged in a hierarchy.
- Graphs can be used to represent networks, while variants of trees can be used for sorting and searching, having their nodes arranged in some relative order based on their values.





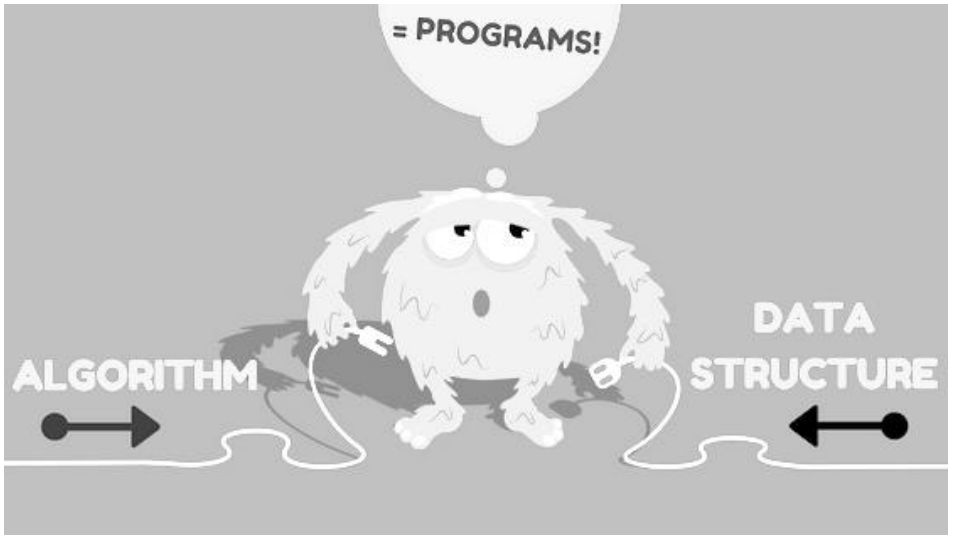
Data Structure Operations

The common operations that can be performed on different data structures are:-

1. **Traversal:** Accessing each data item of data structure in order to process it is called traversal.
2. **Searching:** Finding the location of a given data item.
3. **Insertion:** Adding new data item to the data structure.
4. **Deletion**: Removing an existing data item from a data structure.
5. **Sorting:** Arranging the data items in some logical order(Ascending or descending order).
6. **Merging:** Combining the elements of two similar sorted data structures into a single structure.

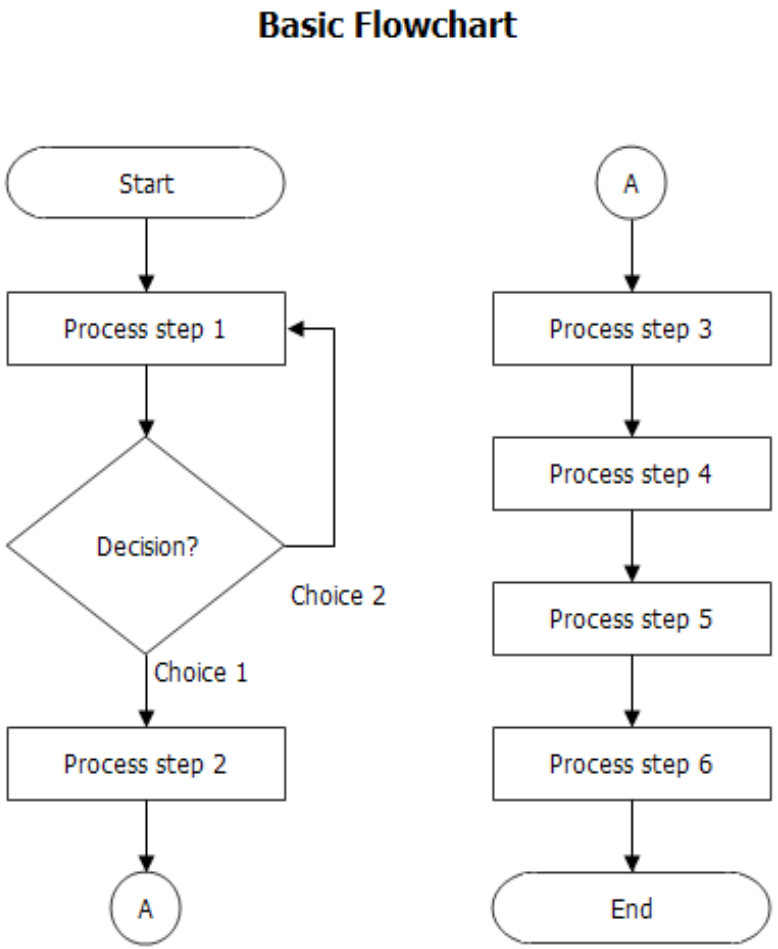
Planning a Computer Program (Revision)

- As a programmer you are not suppose to start directly by coding.
- The most important part of programming is brain storming on how to solve the problem.
  - First step can be on paper.
  - Technically we term such steps as **Pseudocode**.
  - Some programmers also use Algorithm to solve the issue on paper, then start programming.




```
Start
Read Name, Age
IF EOF THEN
   Write "Error Empty File"
ELSE
    DOWHILE not EOF
        Write Name, Age
        Read Name, Age
    ENDDO
ENDIF
Stop
```

Algorithm, Pseudocode and Flow chart (Revision)



- Algorithms can be described in various ways, from pure mathematical formulas to complex graphs, more times than not, without pseudocode.
- Pseudocode describes how you would implement an algorithm without getting into specific (programming language) details.
- Flow Chart as shown in the diagram can be described as pictorial view of how to solve a problem

Algorithm Specification (Revision)

- An ***algorithm*** is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:

1. ***Input***. There are zero or more quantities that are externally supplied.
2. ***Output***. At least one quantity is produced.
3. ***Definiteness***. Each instruction is clear and unambiguous.
4. ***Finiteness***. If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.
5. ***Effectiveness***. Every instruction must be basic enough to be carried out, in principle, by a person using only pencil and paper.

- Example: Algorithm to add two numbers
  - Step 1: Start
  - Step 2: Declare variables num1, num2, result
  - Step 3: Take input from user num1 and num2
  - Step 4: Add num1 and num2 and move the sum to result variable
  - Step 5: Output result
  - Step 6: Stop

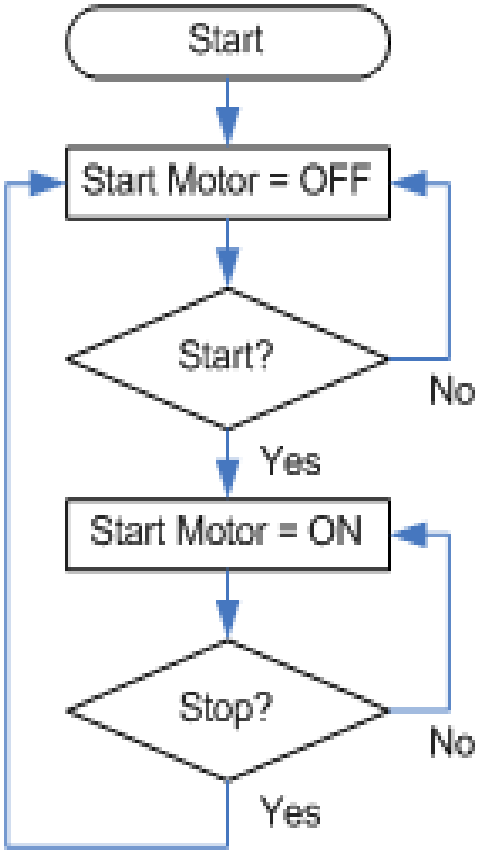
Algorithm Specification (Revision)

- An ***algorithm*** is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:

1. ***Input***. There are zero or more quantities that are externally supplied.
2. ***Output***. At least one quantity is produced.
3. ***Definiteness***. Each instruction is clear and unambiguous.
4. ***Finiteness***. If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.
5. ***Effectiveness***. Every instruction must be basic enough to be carried out, in principle, by a person using only pencil and paper.

- Example: Algorithm to add two numbers
  - Step 1: Start
  - Step 2: Declare variables num1, num2, result
  - Step 3: Take input from user num1 and num2
  - Step 4: Add num1 and num2 and move the sum to result variable
  - Step 5: Output result
  - Step 6: Stop

How Program Solve Problems (Revision)



- **Program Flow Control:** The order in which program statements are executed is called program control flow.
- **Heuristics:** Some problems are very complex or no algorithm exist to solve some problems, at such conditions programmers rely on heuristics.
- e.g. Antivirus programs (practical method, not guaranteed to be optimal, perfect, logical, or rational, but instead sufficient for reaching an immediate goal.)

Why use C to teach Data Structures? (Revision)

- C can implement data structures swiftly and can facilitating faster computations in programs.
- C has many D.S like array, stack, queue, linked list, tree, etc.

C is a low Level Language: C provides access to machine-level concepts (Bytes & Address for example) that other languages try to hide.

C is a small Language: C provides limited set of features than many languages.

C is a permissive Language: C assumes that you know what you're doing, so it allows you a wider degree of flexibility than many languages.

C-Language (Revision)

```
#include <stdio.h>

void main()
{
clrscr();
printf(“Hi Class”)
getch();
}
```

```
#include <stdio.h>

int main()
{
printf("Hi Class");
return 0;
}
```

C is called structured programming language because a program in c language can be divided into small logical functional modules or structures with the help of function procedure

*Structured Programming (*code to be written in a manner that prevents errors.)

Constant , Variables and Keywords (Revision)

- **Constant** is a quantity that doesn't change.

- **Variable** can be considered as a name given to the location in memory where this constant is stored.

$$3X+Y=20$$

Since 3, 20 cannot change they are constant. While X and Y can change hence they are variable.

Constant , Variables and Keywords (Revision)

- Keywords are the words whose meaning has already been explained to the C Compiler.
- The Key word cannot be used as a variable.
- The Key words are also called reserve words.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Data Types

- Data primitives:
  - Integer (int)
  - Character (char)
  - Floating (float)
  - Double (double)

```c
#include <stdio.h>
#include <limits.h>

int main()
{

printf("Size of int : %lu bytes\n", sizeof(int));
printf("Size of char : %lu byte\n", sizeof(char));
printf("Size of float : %lu bytes\n", sizeof(float));
printf("Size of double : %lu bytes\n", sizeof(double));

return 0;
}
```

Sizeof() = This operator allows you to avoid specifying machine-dependent data sizes in your programs

End of Lecture
Any Questions ?
There is no Lab during the first week of the session

- Do try to implement the codes (which are in lectures) by your self to better understand the working of the program