«Computer Science»

Requirements for applicants
by Innopolis University

Contents

Architecture and Organization 2

Digital Logic and Digital Systems 2

Machine Level Representation of Data 2

Assembly Level Machine Organization 3

Memory System Organization and Architecture 3

Interfacing and Communication 3

Networking and Communication 4

Introduction 4

Networked Applications 4

Discrete Structures 4

Basic Logic 4

Graphs and Trees 5

Information Management Concepts 5

Relational Databases 5

Operating Systems 5

Overview of Operating Systems 5

File Systems 6

Programming Languages 6

Fundamental Programming Concepts 6

Fundamental Data Structures 7

Object-Oriented Programming 7

Algorithms and Complexity 8

Basic Analysis 8

Algorithmic Strategies 8

Fundamental Data Structures and Algorithms 9

Architecture and Organization

Digital Logic and Digital Systems

Topics:

- Overview and history of computer architecture;
- Combinational logic;
- Multiple representations/layers of interpretation (hardware is just another layer).

Applicants should be able to

1. Describe the progression of computer technology components from vacuum tubes to VLSI, from mainframe computer architectures to the organization of warehouse-scale computers. [Familiarity]
2. Comprehend the trend of modern computer architectures towards multi-core and that parallelism is inherent in all hardware systems. [Familiarity]
3. Articulate that there are many equivalent representations of computer functionality, including logical expressions and gates, and be able to use mathematical expressions to describe the functions of simple combinational circuits. [Familiarity]

Machine Level Representation of Data

Topics:

- Bits, bytes, and words;
- Numeric data representation and number bases;
- Fixed- and floating-point systems;
- Signed and twos-complement representations;
- Representation of non-numeric data (character codes, graphical data);
- Representation of records and arrays.

Applicants should be able to

1. Explain why everything is data, including instructions, in computers. [Familiarity]
2. Explain the reasons for using alternative formats to represent numerical data. [Familiarity]
3. Describe how negative integers are stored in sign-magnitude and twos-complement representations. [Familiarity]
4. Explain how fixed-length number representations affect accuracy and precision. [Familiarity]
5. Describe the internal representation of non-numeric data, such as characters, strings, records, and arrays. [Familiarity]
6. Convert numerical data from one format to another. [Usage]
7. Write simple programs for string processing and manipulation. [Usage]

Assembly Level Machine Organization

Topics:

- Basic organization of the von Neumann machine;
- Control unit; instruction fetch, decode, and execution;
- Instruction sets and types (data manipulation, control, I/O).

Applicants should be able to

1. Explain the organization of the classical von Neumann machine and its major functional units. [Familiarity]
2. Describe how an instruction is executed in a classical von Neumann machine. [Familiarity]

Memory System Organization and Architecture

Topics:

- Storage systems and their technology;
- Memory hierarchy: importance of temporal and spatial locality;
- Main memory organization and operations;
- Latency, cycle time, bandwidth, and interleaving;
- Cache memories (address mapping, block size, replacement and store policy).

Applicants should be able to

1. Identify the main types of memory technology (e.g., SRAM, DRAM, Flash, magnetic disk) and their relative cost and performance. [Familiarity]
2. Explain the effect of memory latency on running time. [Familiarity]
3. Describe how the use of memory hierarchy (cache, virtual memory) is used to reduce the effective memory latency. [Familiarity]
4. Describe the principles of memory management. [Familiarity]

Interfacing and Communication

Topics:

- I/O fundamentals: handshaking, buffering, programmed I/O, interrupt-driven I/O;
- External storage, physical organization, and drives;
- Buses: bus protocols, arbitration, direct-memory access (DMA);
- Introduction to networks: communications networks as another layer of remote access;
- Multimedia support.

Applicants should be able to

1. Explain how interrupts are used to implement I/O control and data transfers. [Familiarity]
2. Describe data access from a magnetic disk drive. [Familiarity]
3. Compare common network organizations, such as Ethernet/bus, ring, switched vs. routed. [Familiarity]
4. Identify the cross-layer interfaces needed for multimedia access and presentation, from image fetch from remote storage, through transport over a communications network, to staging into local memory, and final presentation to a graphical display. [Familiarity]

Networking and Communication

Introduction

Topics:

- Organization of the Internet (Internet Service Providers, Content Providers, etc.);
- Switching techniques (e.g., circuit, packet);
- Physical pieces of a network, including hosts, routers, switches, ISPs, wireless, LAN, access point, and firewalls;
- Layering principles (encapsulation, multiplexing).

Applicants should be able to

1. Articulate the organization of the Internet. [Familiarity]
2. List and define the appropriate network terminology. [Familiarity]
3. Describe the layered structure of a typical networked architecture. [Familiarity]

Networked Applications

Topics:

- Naming and address schemes (DNS, IP addresses.);
- Distributed applications (client/server, peer-to-peer, cloud, etc.);
- HTTP as an application layer protocol.

Applicants should be able to

1. List the differences and the relations between names and addresses in a network. [Familiarity]
2. Define the principles behind naming schemes and resource location. [Familiarity]

Discrete Structures

Basic Logic

Topics:

- Propositional logic;
- Truth tables;
- Predicate logic.

Applicants should be able to

1. Convert logical statements from informal language to propositional and predicate logic expressions. [Usage]
2. Apply formal methods of symbolic propositional and predicate logic, such as calculating validity of formulae and computing normal forms. [Usage]

Graphs and Trees

Topics:

- Trees;
- Undirected graphs;
- Directed graphs;
- Weighted graphs.

Applicants should be able to

1. Illustrate by example the basic terminology of graph theory, as well as some of the properties and special cases of each type of graph/tree. [Familiarity]
2. Demonstrate different traversal methods for trees and graphs, including pre-, post-, and in-order traversal of trees. [Usage]
3. Model a variety of real-world problems in computer science using appropriate forms of graphs and trees, such as representing a network topology or the organization of a hierarchical file system. [Usage]

Information Management Concepts

Relational Databases

Topics:

- Mapping conceptual schema to a relational schema;
- Entity and referential integrity.

Applicants should be able to

1. Prepare a relational schema from a conceptual model developed using the entity- relationship model. [Usage]
2. Explain and demonstrate the concepts of entity integrity constraint and referential integrity constraint [Usage]

Operating Systems

Overview of Operating Systems

Topics:

- Role and purpose of the operating system;
- Functionality of a typical operating system;
- Mechanisms to support client-server models, hand-held devices;
- Design issues (efficiency, robustness, flexibility, portability, security, compatibility);
- Influences of security, networking, multimedia, windowing systems.

Applicants should be able to

1. Explain the objectives and functions of modern operating systems. [Usage]
2. Identify potential threats to operating systems and the security features design to guard against them. [Familiarity]

File Systems

Topics:

- Files: data, metadata, operations, organization;
- Directories: contents and structure;
- File systems.

Applicants should be able to

1. Describe the choices to be made in designing file systems. [Familiarity]
2. Compare and contrast different approaches to file organization, recognizing the strengths and weaknesses of each. [Usage]

Programming Languages

Fundamental Programming Concepts

Topics:

- Basic syntax and semantics of a higher-level language;
- Variables and primitive data types (e.g., numbers, characters, Booleans);
- Expressions and assignments;
- Simple I/O including file I/O;
- Conditional and iterative control structures;
- Functions and parameter passing;
- The concept of recursion.

Applicants should be able to

1. Analyze and explain the behavior of simple programs involving the fundamental programming constructs variables, expressions, assignments, I/O, control constructs, functions, parameter passing, and recursion. [Familiarity]
2. Identify and describe uses of primitive data types. [Familiarity]
3. Write programs that use primitive data types. [Usage]
4. Modify and expand short programs that use standard conditional and iterative control structures and functions. [Usage]
5. Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, the definition of functions, and parameter passing. [Usage]
6. Write a program that uses file I/O to provide persistence across multiple executions. [Usage]
7. Choose appropriate conditional and iteration constructs for a given programming task. [Usage]
8. Describe the concept of recursion and give examples of its use. [Familiarity]
9. Identify the base case and the general case of a recursively-defined problem. [Familiarity]

Fundamental Data Structures

Topics:

- Arrays;
- Records/structs (heterogeneous aggregates);
- Strings and string processing;
- Abstract data types and their implementation
    - Stacks;
    - Queues;
    - Priority queues;
    - Sets;
    - Maps;
- References and aliasing;
- Linked lists;
- Strategies for choosing the appropriate data structure.

Applicants should be able to

1. Discuss the appropriate use of built-in data structures. [Familiarity]
2. Describe common applications for each of the following data structures: stack, queue, priority queue, set, and map. [Familiarity]
3. Write programs that use each of the following data structures: arrays, records/structs, strings, linked lists, stacks, queues, sets, and maps. [Usage]
4. Compare alternative implementations of data structures with respect to performance. [Familiarity]
5. Describe how references allow for objects to be accessed in multiple ways. [Familiarity]
6. Compare and contrast the costs and benefits of dynamic and static data structure implementations. [Familiarity]
7. Choose the appropriate data structure for modeling a given problem. [Familiarity]

Object-Oriented Programming

Topics:

- Object-oriented design;
- Decomposition into objects carrying state and having behavior;
- Class-hierarchy design for modeling;
- Definition of classes: fields, methods, and constructors;
- Subclasses, inheritance, and method overriding;
- Dynamic dispatch: definition of method-call;

Applicants should be able to

1. Design and implement a class. [Usage]
2. Use subclassing to design simple class hierarchies that allow code to be reused for distinct subclasses. [Usage]
3. Correctly reason about control flow in a program using dynamic dispatch. [Usage]

Algorithms and Complexity

Basic Analysis

Topics:

- Differences among best, expected, and worst case behaviors of an algorithm;
- Asymptotic analysis of upper and expected complexity bounds;
- Big O notation: formal definition;
- Complexity classes, such as constant, logarithmic, linear, quadratic, and exponential;
- Empirical measurements of performance;
- Time and space trade-offs in algorithms.

Applicants should be able to

1. Explain what is meant by “best”, “expected”, and “worst” case behavior of an algorithm. [Familiarity]
2. In the context of specific algorithms, identify the characteristics of data and/or other conditions or assumptions that lead to different behaviors. [Usage]
3. Determine informally the time and space complexity of simple algorithms. [Usage]
4. State the formal definition of big O. [Familiarity]
5. List and contrast standard complexity classes. [Familiarity]
6. Perform empirical studies to validate hypotheses about runtime stemming from mathematical analysis. Run algorithms on input of various sizes and compare performance. [Familiarity]
7. Give examples that illustrate time-space trade-offs of algorithms. [Familiarity]

Algorithmic Strategies

Topics:

- Brute-force algorithms;
- Greedy algorithms;
- Divide-and-conquer;
- Recursive backtracking;
- Dynamic Programming.

Applicants should be able to

1. For each of the strategies (brute-force, greedy, divide-and-conquer, recursive backtracking, and dynamic programming), identify a practical example to which it would apply. [Familiarity]
2. Use a greedy approach to solve an appropriate problem and determine if the greedy rule chosen leads to an optimal solution. [Usage]
3. Use a divide-and-conquer algorithm to solve an appropriate problem. [Usage]
4. Use recursive backtracking to solve a problem such as navigating a maze. [Usage]
5. Use dynamic programming to solve an appropriate problem. [Usage]
6. Determine an appropriate algorithmic approach to a problem. [Usage]

Fundamental Data Structures and Algorithms

Topics:

- Simple numerical algorithms, such as computing the average of a list of numbers, finding the min, max, and mode in a list, approximating the square root of a number, or finding the greatest common divisor;
- Sequential and binary search algorithms;
- Worst case quadratic sorting algorithms (selection, insertion);
- Worst or average case O (N log N) sorting algorithms (quicksort, heapsort, mergesort);
- Hash tables, including strategies for avoiding and resolving collisions;
- Binary search trees
    - Common operations on binary search trees such as select min, max, insert, delete, iterate over tree;
- Graphs and graph algorithms
    - Representations of graphs (e.g., adjacency list, adjacency matrix);
    - Depth- and breadth-first traversals.

Applicants should be able to

1. Implement basic numerical algorithms. [Usage]
2. Implement simple search algorithms and explain the differences in their time complexities. [Usage]
3. Be able to implement common quadratic and O (N log N) sorting algorithms. [Usage]
4. Describe the implementation of hash tables, including collision avoidance and resolution. [Familiarity]
5. Discuss the runtime and memory efficiency of principal algorithms for sorting, searching, and hashing. [Familiarity]
6. Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data. [Familiarity]
7. Explain how tree balance affects the efficiency of various binary search tree operations. [Familiarity]
8. Solve problems using fundamental graph algorithms, including depth-first and breadth-first search. [Usage]
9. Demonstrate the ability to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in a particular context. [Usage]