

### Data Structures

Elementary or Scalar	Built-in data types supported by hardware	int, float, double, char
Structure	Groups multiple scalars, accessed via references	arrays, records, structs
Abstract	Developer-defined data types	enumerations, type declarations
Arrays	a finite, ordered collection of homogeneous elements, each accessed through a subscript value	
Base Type	type of elements or components of the array	
Index	type of the values used to access the individual elements of the array	

### Structures

Structures			
Concept	Description	Syntax	Example
Structure Declaration	A user-defined data type grouping different variables.	struct StructName { datatype var1; datatype var2; };	struct Student { char name[50]; int age; float grade; };
Structure Initialization	Assign values to structure members.	StructName variable = {values};	Student s1 = {"John", 20, 89.5};
Accessing Structure Members	Use dot operator (.) to access members.	variable.member;	cout << s1.name;
Structures with Arrays	Structures containing arrays.	struct StructName { datatype var[size]; };	struct Student { int Grades[5]; } Students[5];
Array of Structures	Using an array to store multiple structures.	StructName arrayName[size];	Student students[10];

### Pointers

Pointers			
Concept	Description	Syntax	Example
Pointer Declaration	A pointer stores the memory address of another variable.	datatype *pointer_name;	int *p;
Reference Operator (&)	Returns the memory address of a variable.	pointer variable = &variable;	int x = 25; int *p = &x;
Dereference Operator (*)	Accesses the value at a pointer's memory address.	cout << *pointer_name;	cout << *p; // Outputs value of x
Pointer Arithmetic	Allows manipulation of memory addresses.	ptr++, ptr--, ptr * n	int arr[3] = {10, 20, 30}; int *p = arr; cout << *(p+1); // 20
Null Pointer	A pointer that doesn't point to any address.	int *p = nullptr;	if (p == nullptr) { cout << "Pointer is null"; }

### Function

Functions			
Concept	Description	Syntax	Example
Function Declaration	Defines a reusable set of statements.	datatype function_name(parameter_list);	int add(int, int);
Function Definition	Specifies function logic.	datatype function_name(parameters) { return value; }	int add(int a, int b) { return a + b; }
Function Call	Executes the function.	function_name(arguments);	int sum = add(5, 3);
Pass by Value	Copies variable values into function parameters.	datatype function_name(datatype var);	int add(int a, int b);
Pass by Reference	Passes the actual memory address.	datatype function_name(datatype &var);	void modify(int &x) { x = 10; }
Function Overloading	Multiple functions with the same name but different parameters.	datatype function_name(params);	int add(int, int); float add(float, float);

### Vectors

Vectors			
Concept	Description	Syntax	Example
Vector Declaration	A dynamic array that can change size.	vector<datatype> vector_name;	vector<int> v;
Vector Initialization	Assign values to vector elements.	vector<datatype> vector_name = {values};	vector<int> v = {1, 2, 3, 4, 5};
Accessing Elements	Retrieve vector elements via index.	vector_name[index];	cout << v[2]; // Output: 3
Adding Elements	Adds an element to the end of the vector.	vector_name.push_back(value);	v.push_back(6);
Removing Elements	Removes the last element.	vector_name.pop_back();	v.pop_back();
Getting Vector Size	Returns the number of elements.	vector_name.size();	int size = v.size();

### Arrays Ex

```
#include <iostream>
using namespace std;
int main() {
    int arr[] = {10, 20, 30, 40, 50};
    for (int i = 0; i < 5; i++) {
        cout << " Element at index " << i
        << ": " << arr[i] << endl;
    }
    return 0;
}
```

### Functions Ex

```
#include <iostream>
using namespace std;
int add(int a, int b) {
    return a + b;
}
int main() {
    int result = add(5, 7);
    cout << "Sum: " << result;
    return 0;
}
```

### Pointers

Pointers	data type that "points" to another value stored in memory	<code>datatype *variable_name;</code>
Reference Pointer	returns the variable's address	<code>pointer_variable = &amp;variable;</code>
Dereference Pointer	returns the value stored in a memory address	<code>pointer_variable = &amp;variable;</code>

### Arrays

Arrays			
Concept	Description	Syntax	Example
Array Declaration	Collection of elements stored in contiguous memory.	<code>datatype array_name[array_size];</code>	<code>int numbers[5];</code>
Array Initialization	Assign values to array elements.	<code>datatype array_name[size] = {values};</code>	<code>int numbers[5] = {1, 2, 3, 4, 5};</code>
Accessing Elements	Retrieve array elements via index.	<code>array_name[index];</code>	<code>cout &lt;&lt; numbers[2]; // Output: 3</code>
Multidimensional Arrays	Arrays with multiple indices.	<code>datatype array_name[x][y];</code>	<code>int matrix[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};</code>
Sorting (Selection Sort)	Arrange elements in order.	<code>for (int i = 0; i &lt; n-1; i++) { ... swap(arr[i], arr[minIndex]); }</code>	Selection Sort Algorithm
Searching (Binary Search)	Finds an element in a sorted array.	<code>int binarySearch(int arr[], int left, int right, int x);</code>	<code>binarySearch(arr, 0, n-1, key);</code>

### Basic Operation

Extraction	<code>X = arr[3]; // Gets the 4th element</code>
Storing	<code>arr[4] = 100; // Stores 100 at index 4</code>
Copying	<code>for (int i = 0; i &lt; size; i++) { newArr[i] = oldArr[i]; }</code>

### Sorting Algorithm

```
for (int i = 0; i < size - 1; i++) {
    int minIndex = i;
    for (int j = i + 1; j < size; j++) {
        if (arr[j] < arr[minIndex]) {
            minIndex = j;
        }
    }
    swap(arr[i], arr[minIndex]);
}
```

### Bubble Sort

```
for (int i = 0; i < size - 1; i++) {
    for (int j = 0; j < size - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
            swap(arr[j], arr[j + 1]);
        }
    }
}
```

### Vectors Ex

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v = {10, 20, 30};
    v.push_back(40);
    for (int i = 0; i < v.size(); i++) {
        cout << "Vector element at index "
        << i << ": " << v[i] << endl;
    }
    return 0;
}
```

### Structures Ex

```
#include <iostream>
using namespace std;
struct Student {
    char name[50];
    int age;
    float grade;
};
int main() {
    Student s1 = {"John", 20, 89.5};
    cout << " Name: " << s1.name << " Age: " <<
    s1.age << " Grade: " << s1.grade;
    return 0;
}
```