Class Note for Session 19: Advanced Data Types and Sorting

Introduction

Session 19 of the "Elementary Programming in C" course focuses on advanced data types, specifically **structures**, and introduces two fundamental sorting algorithms: **Insertion Sort** and **Bubble Sort**. This session builds on previous knowledge of arrays, pointers, and functions, enabling students to create complex data structures and perform efficient data manipulation.

---

1. Need for Structures

Explanation

Structures in C allow grouping of variables of different data types under a single name, enabling the creation of user-defined data types. They are useful for representing real-world entities (e.g., a student with a name, ID, and marks) where multiple attributes need to be stored together. Unlike arrays, which store elements of the same type, structures can hold heterogeneous data.

Example

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

// Defining a structure for a student
struct Student {
    char name[50];
    int id;
    float marks;
};

int main() {
    // Declaring a structure variable
    struct Student s1;

    // Assigning values
    strcpy(s1.name, "John Doe");
    s1.id = 101;
    s1.marks = 85.5;

    // Accessing and printing structure members
    printf("Student Name: %s\n", s1.name);
    printf("Student ID: %d\n", s1.id);
    printf("Student Marks: %.2f\n", s1.marks);

    return 0;
}
```

Output:

```
Student Name: John Doe
Student ID: 101
Student Marks: 85.50
```

Classwork

1. Create a structure called `Book` with fields for `title` (string), `author` (string), and `price`

(float). Declare a variable of this structure, assign values, and print them.
2. Modify the above program to take user input for the `Book` structure fields using `scanf()` and print the details.

---

2. Defining Structures

Explanation

A structure is defined using the `struct` keyword followed by the structure name and a block containing member declarations. The structure definition acts as a blueprint, and variables of that structure type can be created to store actual data.

Example

```c
#include <stdio.h>

// Defining a structure
struct Point {
    int x;
    int y;
};

int main() {
    // Declaring structure variables
    struct Point p1, p2;

    // Assigning values
    p1.x = 10;
    p1.y = 20;
    p2.x = 30;
    p2.y = 40;

    // Printing coordinates
    printf("Point 1: (%d, %d)\n", p1.x, p1.y);
    printf("Point 2: (%d, %d)\n", p2.x, p2.y);

    return 0;
}
```

Output:

```
Point 1: (10, 20)
Point 2: (30, 40)
```

Classwork

1. Define a structure `Employee` with fields `name` (string), `emp_id` (int), and `salary` (float). Create two variables of this structure and initialize them with different values. Print the details of both employees.
2. Write a program to calculate the distance between two points using the `Point` structure

from the example. Use the formula: `distance = sqrt((x2-x1)^2 + (y2-y1)^2)`.

---

3. Assignment of Values to Structure Elements

Explanation

Structure elements are accessed using the dot operator (`.`) with a structure variable. Values can be assigned directly or through user input. Initialization can also be done at the time of declaration using curly braces `{}`.

Example

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

struct Car {
    char model[20];
    int year;
    float price;
};

int main() {
    // Initializing structure at declaration
    struct Car car1 = {"Toyota", 2020, 25000.50};

    // Assigning values individually
    struct Car car2;
    strcpy(car2.model, "Honda");
    car2.year = 2018;
    car2.price = 18000.75;

    // Printing details
    printf("Car 1: %s, %d, $%.2f\n", car1.model, car1.year, car1.price);
    printf("Car 2: %s, %d, $%.2f\n", car2.model, car2.year, car2.price);

    return 0;
}
```

Output:

```
Car 1: Toyota, 2020, $25000.50
Car 2: Honda, 2018, $18000.75
```

Classwork

1. Create a structure `Rectangle` with fields `length` and `width`. Initialize one instance at declaration and another by assigning values individually. Print both rectangles' details.
2. Write a program to take user input for a `Car` structure and display the entered details.

4. Accessing Structure Elements

Explanation

Structure elements are accessed using the dot operator (`.`) for structure variables or the arrow operator (`->`) for pointers to structures. This allows reading or modifying individual members of a structure.

Example

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

struct Person {
    char name[30];
    int age;
};

int main() {
    struct Person p1;
    struct Person *ptr = &p1;

    // Accessing using dot operator
    strcpy(p1.name, "Alice");
    p1.age = 25;

    // Accessing using arrow operator
    printf("Using dot: %s, %d\n", p1.name, p1.age);
    printf("Using arrow: %s, %d\n", ptr->name, ptr->age);

    return 0;
}
```

Output:

```
Using dot: Alice, 25
Using arrow: Alice, 25
```

Classwork

1. Create a structure `Product` with fields `name` (string), `id` (int), and `price` (float). Use a pointer to the structure to assign and access values, then print them.
2. Modify the above program to update the `price` of the `Product` using the pointer and display the updated details.

---

5. Passing Structures as Function Arguments

Explanation

Structures can be passed to functions either by value (copy of the structure) or by reference (using a pointer). Passing by reference is more efficient for large structures as it avoids copying.

Example

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

struct Student {
    char name[50];
    int roll_no;
};

void displayStudent(struct Student s) {
    printf("Name: %s, Roll No: %d\n", s.name, s.roll_no);
}

int main() {
    struct Student s1;
    strcpy(s1.name, "Bob");
    s1.roll_no = 102;

    displayStudent(s1); // Passing structure by value

    return 0;
}
```

Output:

```
Name: Bob, Roll No: 102
```

Classwork

1. Write a function that takes a `Rectangle` structure (with `length` and `width`) as an argument and returns its area. Call this function from `main()` and print the result.
2. Modify the above program to pass the `Rectangle` structure by reference (using a pointer) and update its `length` inside the function.

---

6. Arrays of Structures

Explanation

An array of structures allows storing multiple instances of a structure. Each element of the array is a structure variable, enabling efficient storage and manipulation of multiple entities.

Example

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

struct Book {
    char title[50];
    float price;
};

int main() {
    struct Book library[3];

    // Initializing array of structures
    strcpy(library[0].title, "C Programming");
    library[0].price = 29.99;
    strcpy(library[1].title, "Data Structures");
    library[1].price = 39.99;
    strcpy(library[2].title, "Algorithms");
    library[2].price = 49.99;

    // Printing all books
    for (int i = 0; i < 3; i++) {
        printf("Book %d: %s, $%.2f\n", i + 1, library[i].title, library[i].price);
    }

    return 0;
}
```

Output:

```
Book 1: C Programming, $29.99
Book 2: Data Structures, $39.99
Book 3: Algorithms, $49.99
```

Classwork

1. Create an array of 5 `Student` structures with fields `name` and `marks`. Initialize them with different values and print all students' details.
2. Write a program to find the student with the highest marks from an array of `Student` structures.

---

7. Pointers to Structures

Explanation

A pointer to a structure holds the address of a structure variable. The arrow operator (`->`) is used to access structure members through a pointer, making it efficient for manipulating structures.

Example

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

struct Employee {
    char name[30];
    int id;
};

int main() {
    struct Employee emp = {"Charlie", 1001};
    struct Employee *ptr = &emp;

    // Accessing using pointer
    printf("Employee: %s, ID: %d\n", ptr->name, ptr->id);

    // Modifying using pointer
    ptr->id = 1002;
    printf("Updated Employee: %s, ID: %d\n", ptr->name, ptr->id);

    return 0;
}
```

Output:

```
Employee: Charlie, ID: 1001
Updated Employee: Charlie, ID: 1002
```

Classwork

1. Create a structure `Point` with `x` and `y` coordinates. Use a pointer to update the coordinates and print the updated values.
2. Write a program that uses a pointer to a `Book` structure to modify its `price` and display the updated details.

---

8. Passing Pointers to Structures as Function Arguments

Explanation

Passing a pointer to a structure to a function allows the function to modify the original structure without creating a copy. This is memory-efficient and commonly used in C programming.

Example

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

struct Product {
    char name[20];
    float price;
};

void updatePrice(struct Product *p, float new_price) {
    p->price = new_price;
}

int main() {
    struct Product prod = {"Laptop", 999.99};
    printf("Before: %s, $%.2f\n", prod.name, prod.price);

    updatePrice(&prod, 1099.99); // Passing pointer to structure
    printf("After: %s, $%.2f\n", prod.name, prod.price);

    return 0;
}
```

Output:

```
Before: Laptop, $999.99
After: Laptop, $1099.99
```

Classwork

1. Write a function that takes a pointer to a `Student` structure and updates its `marks`. Call this function and print the updated student details.
2. Create a function that takes a pointer to a `Rectangle` structure and updates its `width`. Verify the update in `main()`.

---

9. Use of `typedef`

Explanation

The `typedef` keyword creates an alias for a data type, simplifying structure declarations and improving code readability. It is commonly used with structures to avoid writing `struct` repeatedly.

Example

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

typedef struct {
    char name[30];
    int id;
} Employee;

int main() {
    Employee emp1; // No need to write 'struct'
    strcpy(emp1.name, "David");
    emp1.id = 2001;

    printf("Employee: %s, ID: %d\n", emp1.name, emp1.id);

    return 0;
}
```

Output:

```
Employee: David, ID: 2001
```

Classwork

1. Use `typedef` to create an alias for a `Point` structure with `x` and `y` fields. Declare and initialize a variable using the alias and print its values.
2. Rewrite the `Book` structure example using `typedef` and create an array of 3 books, then print their details.

---

10. Insertion Sort Algorithm

Explanation

Insertion Sort is a simple sorting algorithm that builds the sorted array one element at a time. It iterates through the array, placing each element in its correct position among the previously sorted elements. It is efficient for small datasets.

Example

```c
#include <stdio.h>

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22};
    int n = 5;

    printf("Before sorting: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    insertionSort(arr, n);

    printf("\nAfter sorting: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output:

```
Before sorting: 64 34 25 12 22
After sorting: 12 22 25 34 64
```

Classwork

1. Modify the Insertion Sort program to sort an array of 6 user-input integers.
2. Write a program to sort an array of `Student` structures based on their `marks` using Insertion Sort.

---

11. Bubble Sort Algorithm

Explanation

Bubble Sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The process continues until no swaps are needed, indicating the array is sorted. It is simple but inefficient for large datasets.

Example

```c
#include <stdio.h>

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap elements
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22};
    int n = 5;

    printf("Before sorting: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    bubbleSort(arr, n);

    printf("\nAfter sorting: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output:

```
Before sorting: 64 34 25 12 22
After sorting: 12 22 25 34 64
```

Classwork

1. Modify the Bubble Sort program to sort an array of 5 user-input floating-point numbers.
2. Write a program to sort an array of `Book` structures based on their `price` using Bubble Sort.

General Classwork

1. Create a program that defines a `Library` structure with fields `name`, `book_count`, and an array of `Book` structures (each with `title` and `price`). Initialize the library with 3 books and print all details.
2. Write a program that uses an array of 4 `Employee` structures. Allow the user to input details for each employee, sort them by `salary` using Bubble Sort, and print the sorted list.
3. Create a function that takes a pointer to a `Student` structure array and sorts it by `marks` using Insertion Sort. Test it with 5 students.
4. Write a program that uses `typedef` to define a `Product` structure. Create an array of 3 products, update their prices using a function that takes a pointer, and print the updated details.

Objective Questions

1. **What is the purpose of a structure in C?**
   a) To store elements of the same data type
   b) To group variables of different data types
   c) To define a function
   d) To allocate memory dynamically
   Answer: b) To group variables of different data types

2. **How do you access a structure member using a pointer?**
   a) Using the dot operator (`.`)
   b) Using the arrow operator (`->`)
   c) Using the asterisk operator (`*`)
   d) Using the ampersand operator (`&`)
   Answer: b) Using the arrow operator (`->`)
3. **What does the `typedef` keyword do?**
   a) Defines a new function
   b) Creates an alias for a data type
   c) Allocates memory for a variable
   d) Declares a pointer
   Answer: b) Creates an alias for a data type
4. **In Insertion Sort, what is the role of the `key` variable?**
   a) Stores the smallest element
   b) Holds the current element being inserted
   c) Tracks the number of comparisons
   d) Stores the sorted array
   Answer: b) Holds the current element being inserted
5. **Which sorting algorithm compares adjacent elements and swaps them if they are in the wrong order?**
   a) Insertion Sort
   b) Bubble Sort
   c) Selection Sort
   d) Quick Sort
   Answer: b) Bubble Sort
6. **How can a structure be passed to a function efficiently?**
   a) By value
   b) By reference using a pointer
   c) By copying the entire structure
   d) By using an array
   Answer: b) By reference using a pointer
7. **What is the output of the following code?**

```c
#include <stdio.h>
struct Point { int x, y; };
int main() {
    struct Point p = {5, 10};
    struct Point *ptr = &p;
    printf("%d", ptr->x);
    return 0;
}
```

a) 5
b) 10
c) Error
d) 0
Answer: a) 5

8. **In Bubble Sort, how many passes are required to sort an array of n elements?**
   a) n
   b) n-1
   c) n/2
   d) n^2
   Answer: b) n-1

This class note provides a comprehensive overview of Session 19, with clear explanations, practical examples, and exercises to reinforce learning. Students should practice the classwork and review the objective questions to solidify their understanding.