Module 4.2:
Structures And Unions

Structures And Unions

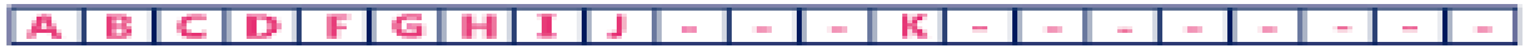
- Introduction, Declaring and defining Structure,
- Structure Initialization,
- Accessing and Displaying Structure Members,
- Array of Structures,
- Introduction to Unions,
- Structure Vs Unions

Arrays

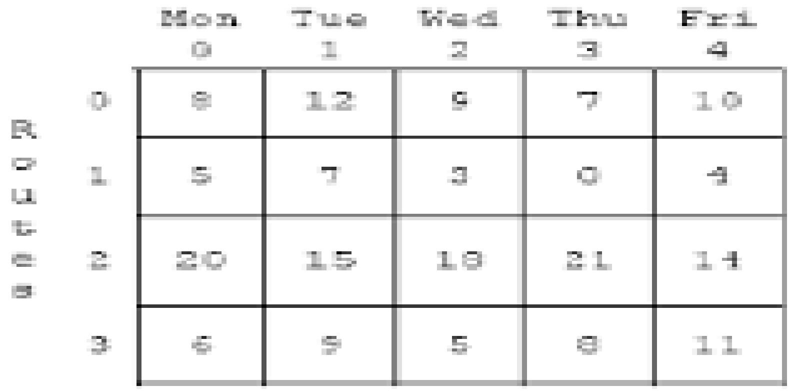
An array is a sequence of data item of homogeneous values (same type).

Arrays are of two types:

1. One dimensional arrays (1D array)
2. Multidimensional arrays (2D, #D, etc.)



1D array



		Mon 0	Tue 1	Wed 2	Thu 3	Fri 4
Routes	0	8	12	9	7	10
	1	5	7	3	0	4
	2	20	15	18	21	14
	3	6	9	5	8	11

2D array

Structures

- Structure is a group of variables of different data types represented by a single name.

- Syntax of struct

```
struct structureName
{
        dataType member1;
        dataType member2;
        ...
};
```

Structures

- Variable declaration of a structure

```
struct structureName
{
        dataType member1;
        dataType member2;
        ...
};
int main()
{
struct  struct_name  var_name;
return 0;
}
```

OR

```
struct struct_name
{
  DataType member1_name;
  DataType member2_name;
  DataType member3_name;
  …
} var_name;
```

Structures

- Accessing members of a structure

```
var_name.member1_name;
var_name.member2_name;
…
```

Structures

- Accessing members of a structure

var_name.member1_name;
var_name.member2_name;
…

- Assign values to structure members

1) Using Dot(.) operator

var_name.memeber_name = value;

2) All members assigned in one statement

struct struct_name var_name =
{value for memeber1, value for memeber2
…so on for all the members}

Structures And Unions

Task

- WAP to display student name, roll number and CGPI using structures.

- WAP to display employee name, ID and year of experience.

- **Write separate code to Assign value at compile time and runtime**

Structures

- Array of Structures

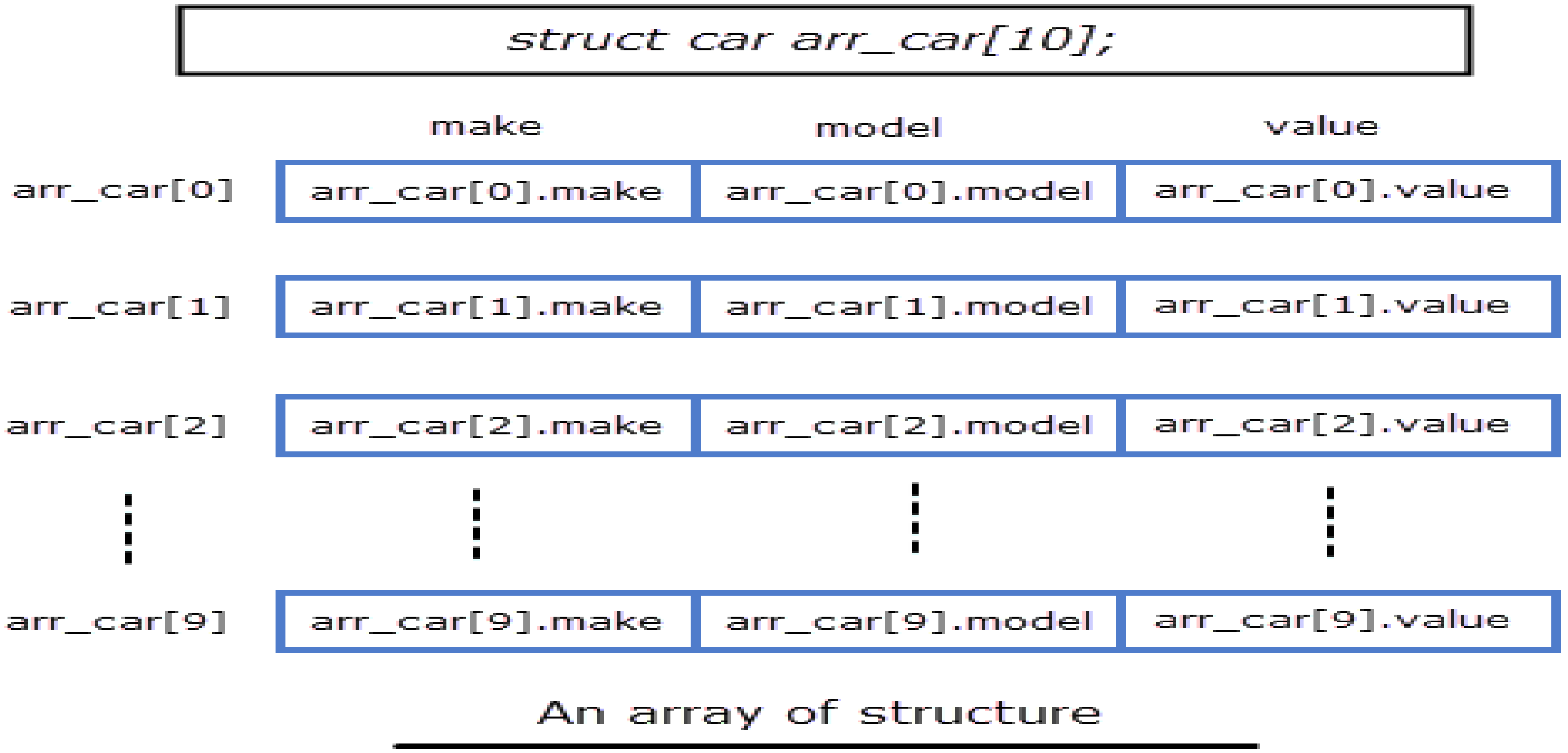
Declaring an array of structure is same as declaring an array of fundamental types. Since an array is a collection of elements of the same type. In an array of structures, each element of an array is of the structure type.

Syntax

```
struct car
{
    char make[20];
    char model[30];
    int year;
};
struct car arr_car[10];
```

Structures

- Array of Structures



An array of structure

Structures

- Array of Structures

Task

- WAP to display car name and number for 5 cars using array of structures

- Define array size
- Write for loop to accept as well as to display elements.

Structures

- Array of Structures

Task

- WAP to display car name and number for 5 cars using array of structures

- Define array size
- Write for loop to accept as well as to display elements.

```
for(i = 0; i < 5; i++ )
{
   cout<< arr_car[i].name<< arr_car[i].number;
}
```

Unions

- Unions
- A union is a special data type available in C that allows to store different data types in the same memory location.

- You can define a union with many members, but only one member can contain a value at any given time.

- Unions provide an efficient way of using the same memory location for multiple-purpose.

Unions

- Unions

```
union [union tag]
{
   member definition;
   member definition;
   ...
   member definition;
} [one or more union variables];
```

Unions

- Unions

```
union [union tag]
{
   member definition;
   member definition;
   ...
   member definition;
} [one or more union variables];
```

```
union Data
{
   int i;
   float f;
   char str[20];
} data;
```

Unions

- Unions

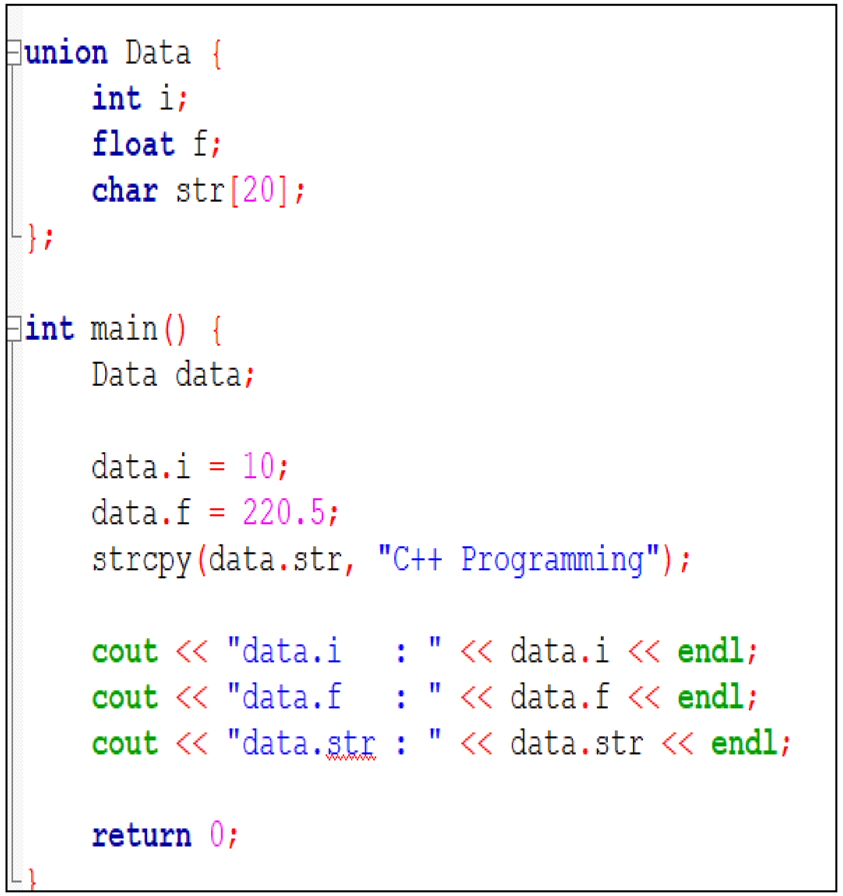
```
union Data
{ int i;
  float f;
  char str[20];
} data;
```

Now, a variable of type Data can store an integer, a floating-point number, or a string of characters. It means a single variable, i.e., same memory location, can be used to store multiple types of data. You can use any built-in or user defined data types inside a union based on your requirement.

The memory occupied by a union will be large enough to hold the largest member of the union. For example, in the above example, Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by a character string.

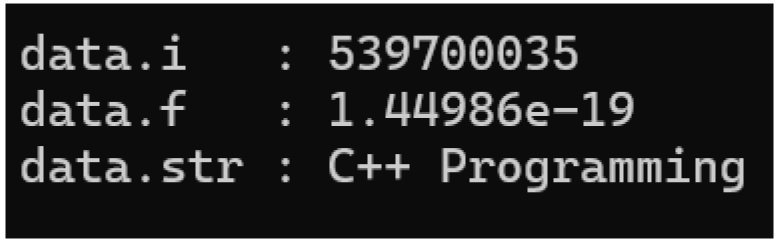
Unions

- Unions



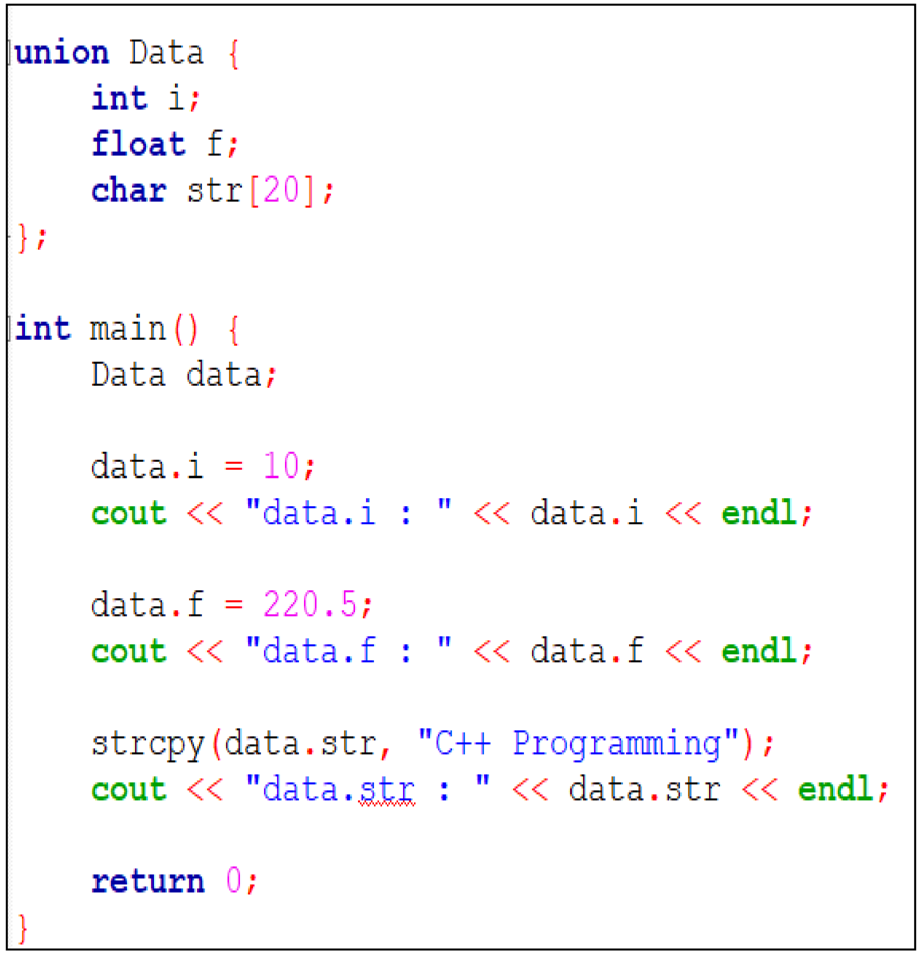
```
union Data {
    int i;
    float f;
    char str[20];
};

int main() {
    Data data;

    data.i = 10;
    data.f = 220.5;
    strcpy(data.str, "C++ Programming");

    cout << "data.i   : " << data.i << endl;
    cout << "data.f   : " << data.f << endl;
    cout << "data.str : " << data.str << endl;

    return 0;
}
```

Here, we can see that the values of i and f members of union got corrupted because the final value assigned to the variable has occupied the memory location and this is the reason that the value of str member is getting printed very well.



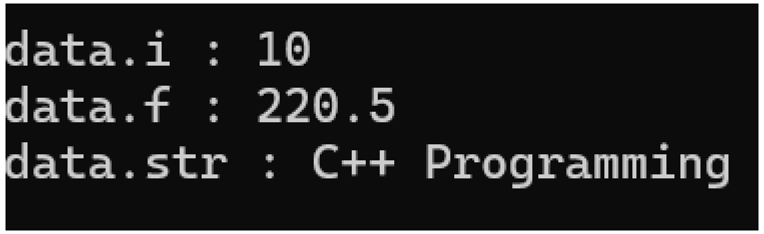
```
data.i   : 539700035
data.f   : 1.44986e-19
data.str : C++ Programming
```

• Unions



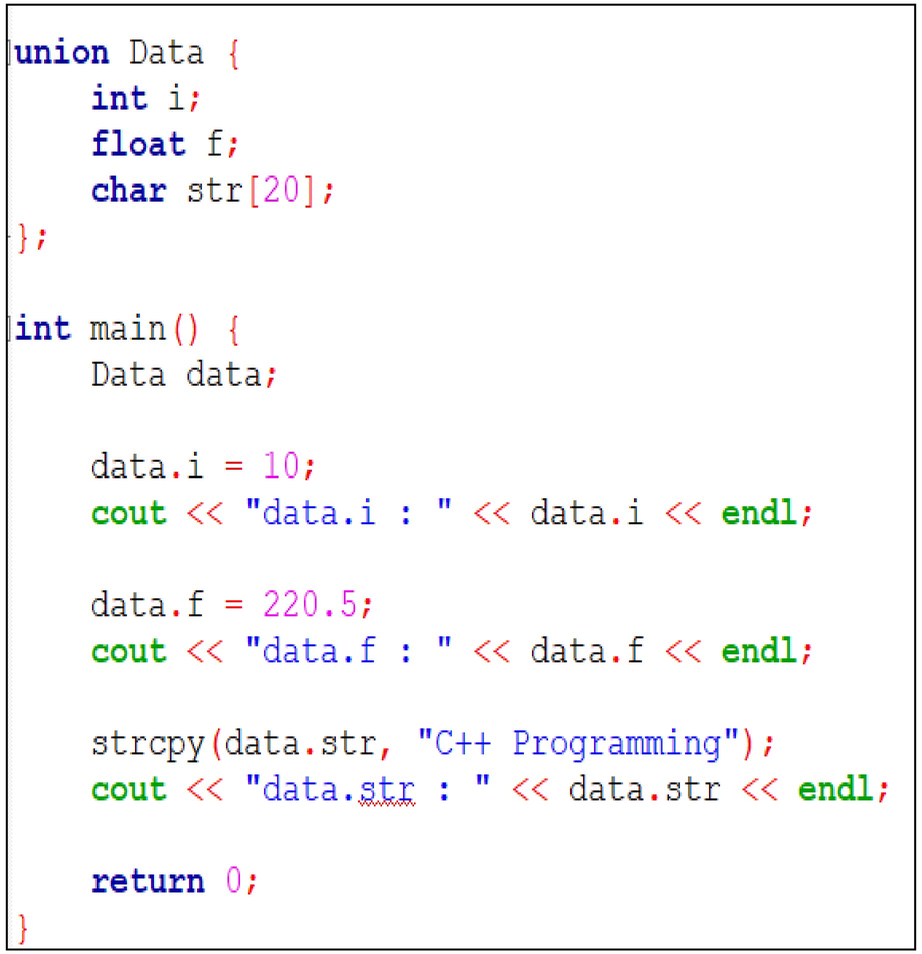
```
union Data {
    int i;
    float f;
    char str[20];
};

int main() {
    Data data;

    data.i = 10;
    cout << "data.i : " << data.i << endl;

    data.f = 220.5;
    cout << "data.f : " << data.f << endl;

    strcpy(data.str, "C++ Programming");
    cout << "data.str : " << data.str << endl;

    return 0;
}
```

Here, all the members are getting printed very well because one member is being used at a time.



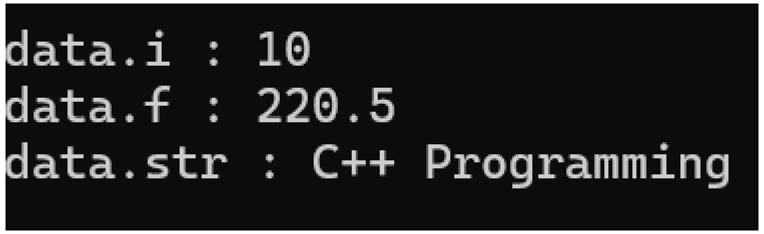
```
data.i : 10
data.f : 220.5
data.str : C++ Programming
```

• Unions



```
union Data {
    int i;
    float f;
    char str[20];
};

int main() {
    Data data;

    data.i = 10;
    cout << "data.i : " << data.i << endl;

    data.f = 220.5;
    cout << "data.f : " << data.f << endl;

    strcpy(data.str, "C++ Programming");
    cout << "data.str : " << data.str << endl;

    return 0;
}
```

Here, all the members are getting printed very well because one member is being used at a time.



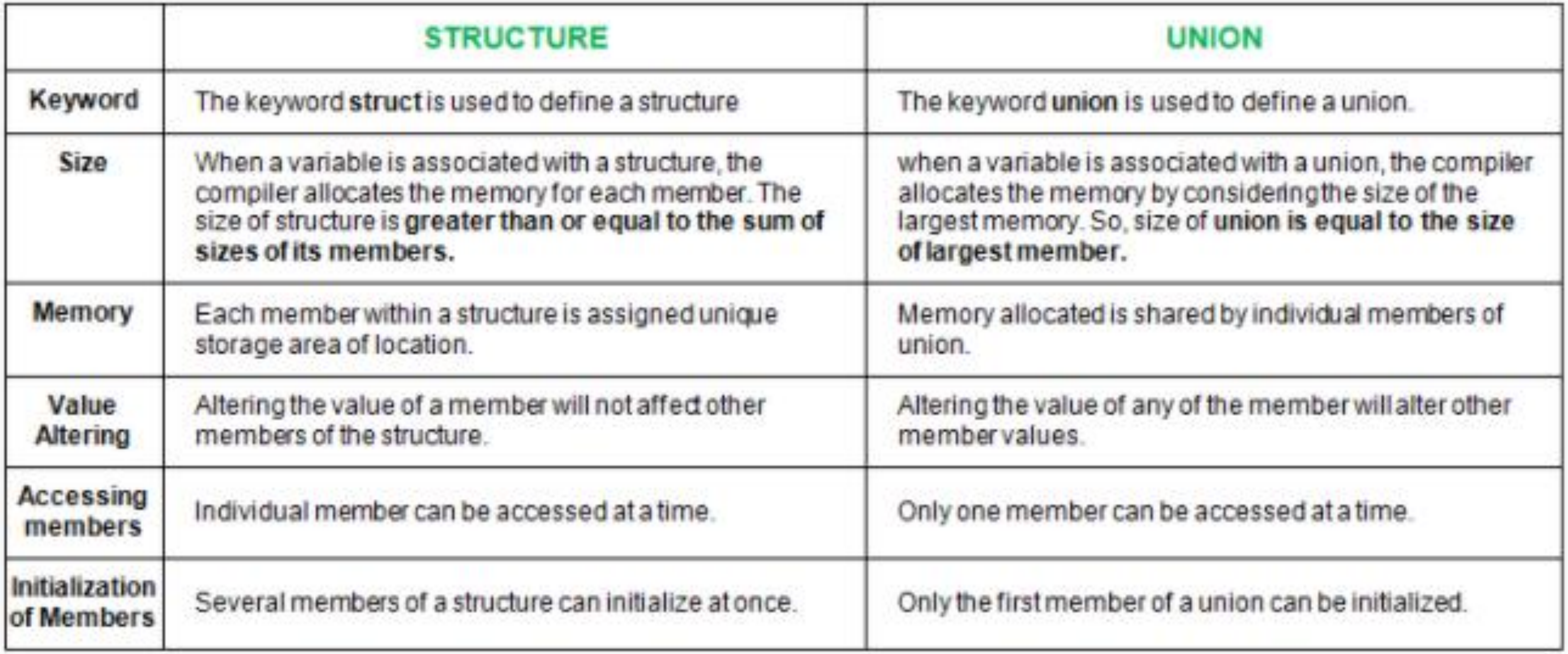
```
data.i : 10
data.f : 220.5
data.str : C++ Programming
```

Structure vs Union

Feature	Structure	Union
Keyword	The keyword 'struct' is used to define a structure.	The keyword 'union' is used to define a union.
Size	When a variable is associated with a structure, memory is allocated for each member. The total size is greater than or equal to the sum of sizes of its members.	When a variable is associated with a union, memory is allocated equal to the size of the largest member.
Memory	Each member has its own unique storage location.	All members share the same memory location.
Value Altering	Changing one member does not affect other members.	Changing one member alters the value of other members.
Initialization	Multiple members can be initialized at once.	Only the first member can be initialized.

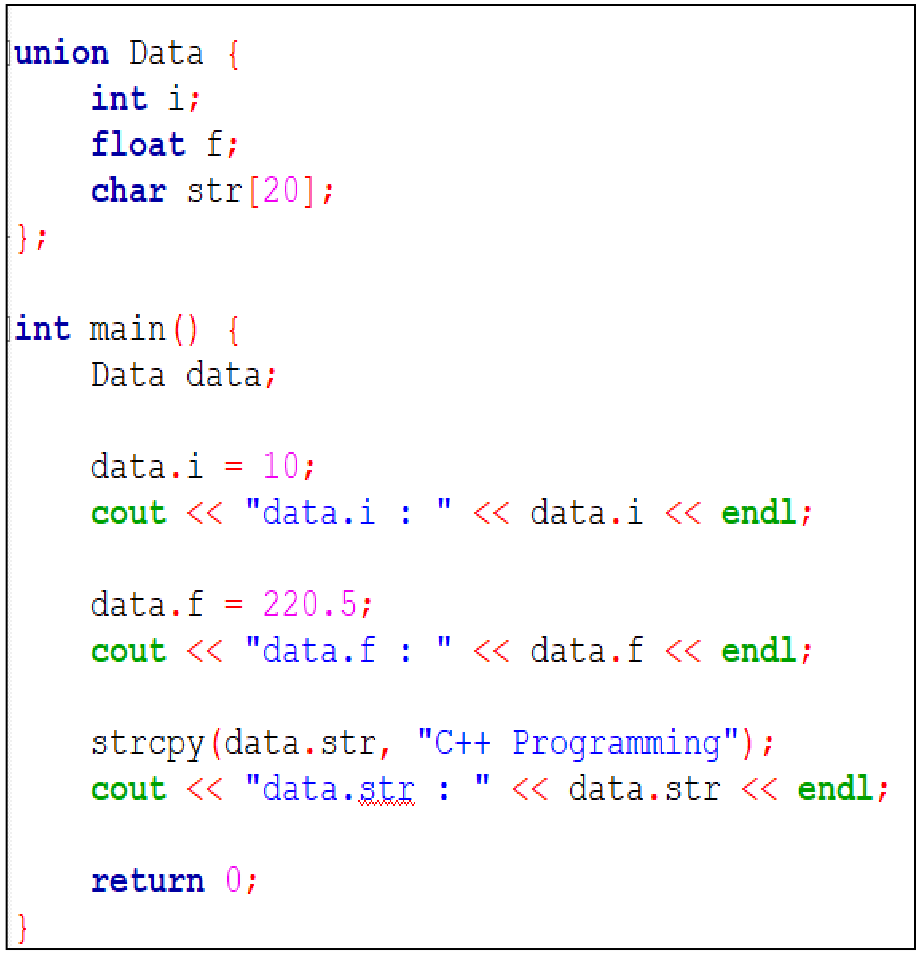
Structures And Unions

- Structures Vs Unions



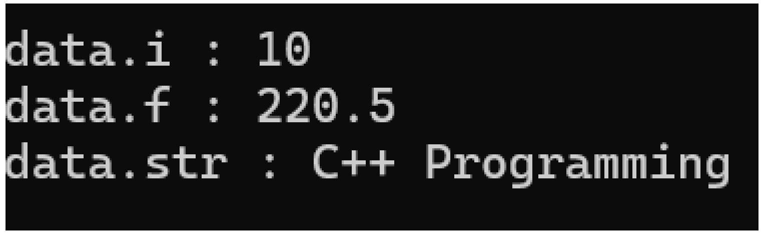
	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

- Unions



```
union Data {
    int i;
    float f;
    char str[20];
};

int main() {
    Data data;

    data.i = 10;
    cout << "data.i : " << data.i << endl;

    data.f = 220.5;
    cout << "data.f : " << data.f << endl;

    strcpy(data.str, "C++ Programming");
    cout << "data.str : " << data.str << endl;

    return 0;
}
```

Here, all the members are getting printed very well because one member is being used at a time.



```
data.i : 10
data.f : 220.5
data.str : C++ Programming
```

Array of structures

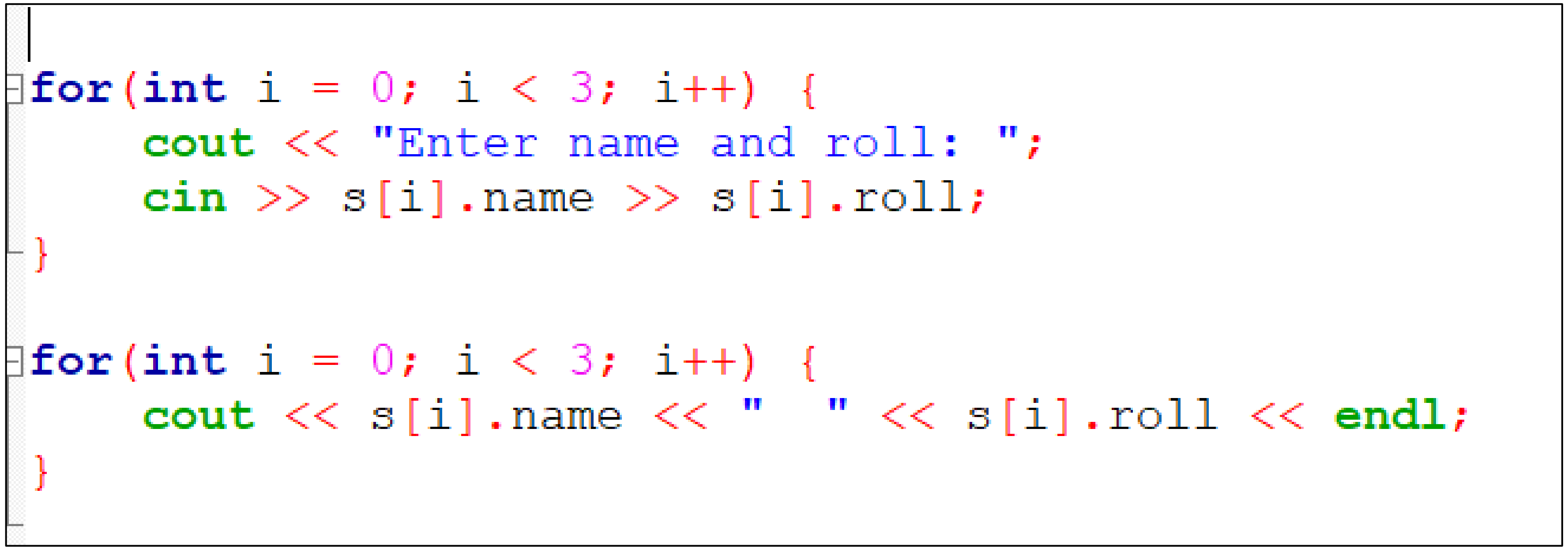
- Combines both an array and a structure to handle complex data.

Declaring array of structures example:

```
struct Student {
   char name[50];
   int roll;
   float marks;
};

// Array of 5 Student records
Student s[5];
```

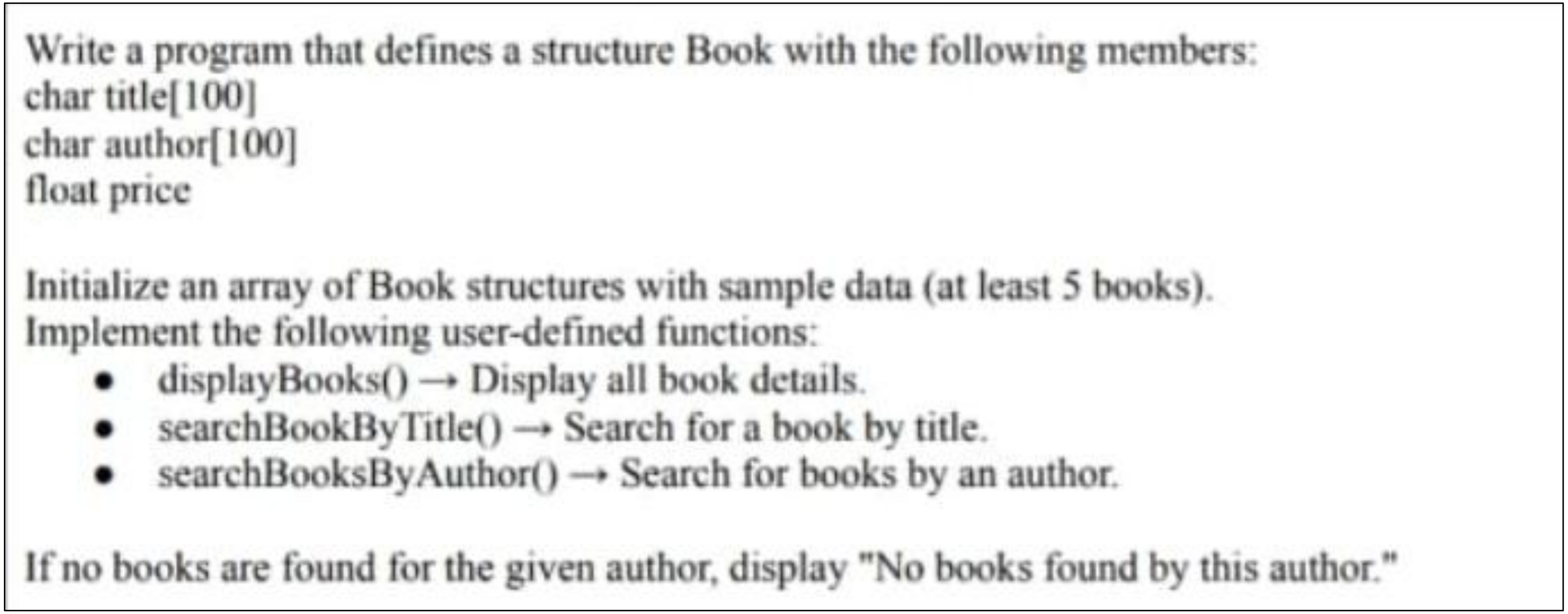
Accessing elements

```
s[0].roll = 101;
cin >> s[1].name;
cout << s[2].marks;
```



```
for(int i = 0; i < 3; i++) {
    cout << "Enter name and roll: ";
    cin >> s[i].name >> s[i].roll;
}

for(int i = 0; i < 3; i++) {
    cout << s[i].name << "  " << s[i].roll << endl;
}
```

Exercise



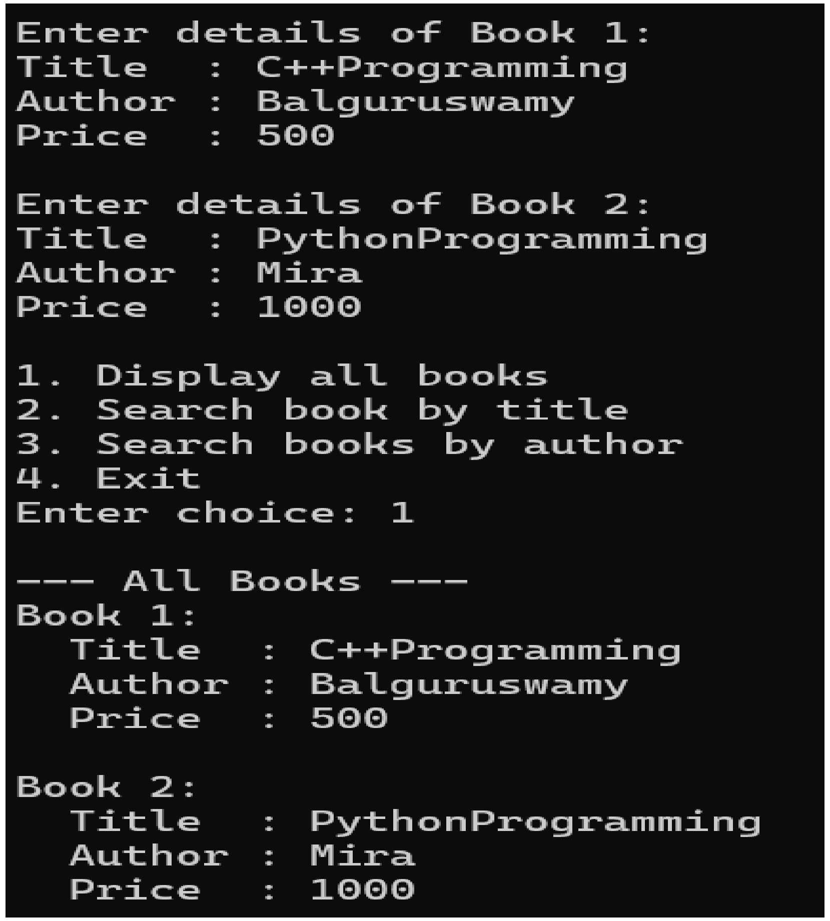
Write a program that defines a structure Book with the following members:
char title[100]
char author[100]
float price

Initialize an array of Book structures with sample data (at least 5 books).
Implement the following user-defined functions:

- displayBooks() → Display all book details.
- searchBookByTitle() → Search for a book by title.
- searchBooksByAuthor() → Search for books by an author.

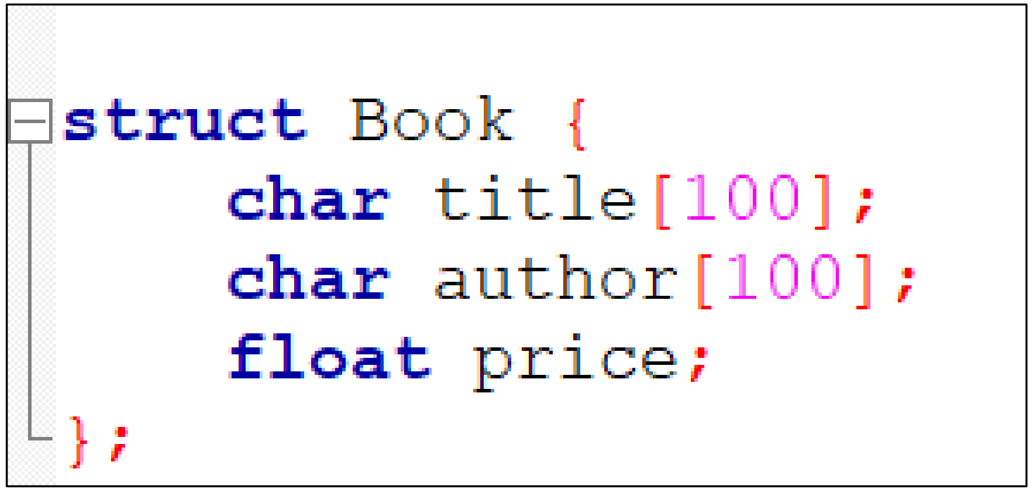
If no books are found for the given author, display "No books found by this author."

Output



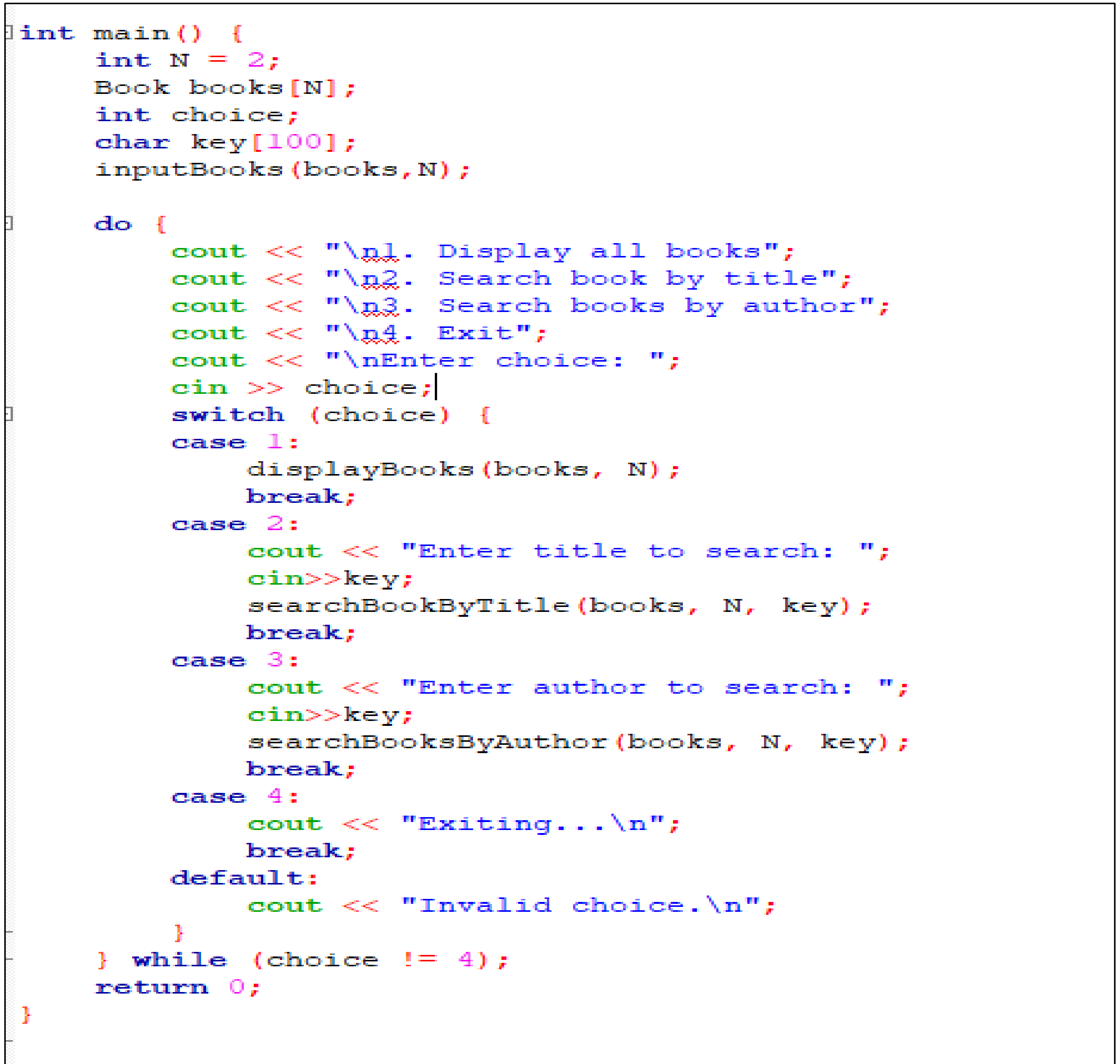
```
Enter details of Book 1:
Title  : C++Programming
Author : Balguruswamy
Price  : 500

Enter details of Book 2:
Title  : PythonProgramming
Author : Mira
Price  : 1000

1. Display all books
2. Search book by title
3. Search books by author
4. Exit
Enter choice: 1

--- All Books ---
Book 1:
  Title  : C++Programming
  Author : Balguruswamy
  Price  : 500

Book 2:
  Title  : PythonProgramming
  Author : Mira
  Price  : 1000
```

Defining structure



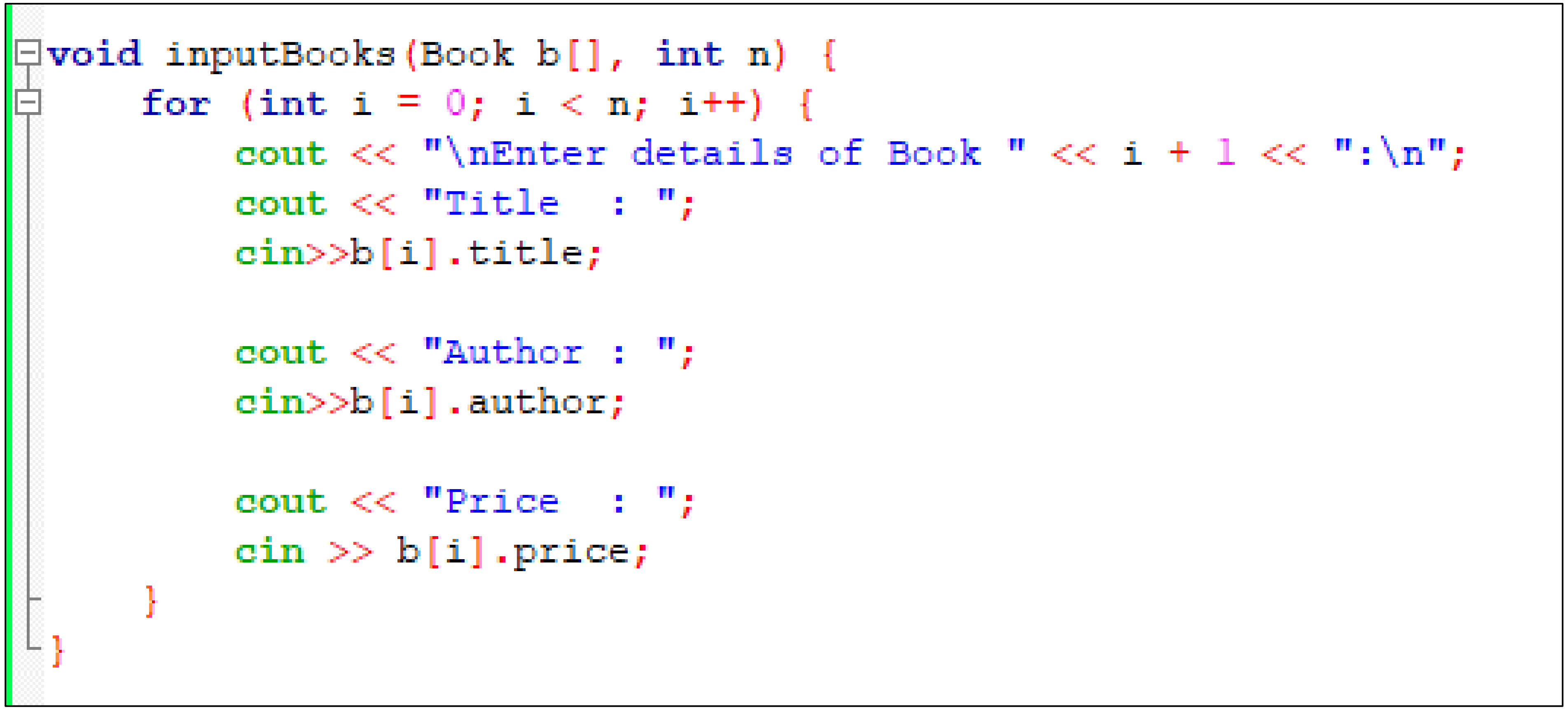
```
struct Book {
	char title[100];
	char author[100];
	float price;
};
```

In main



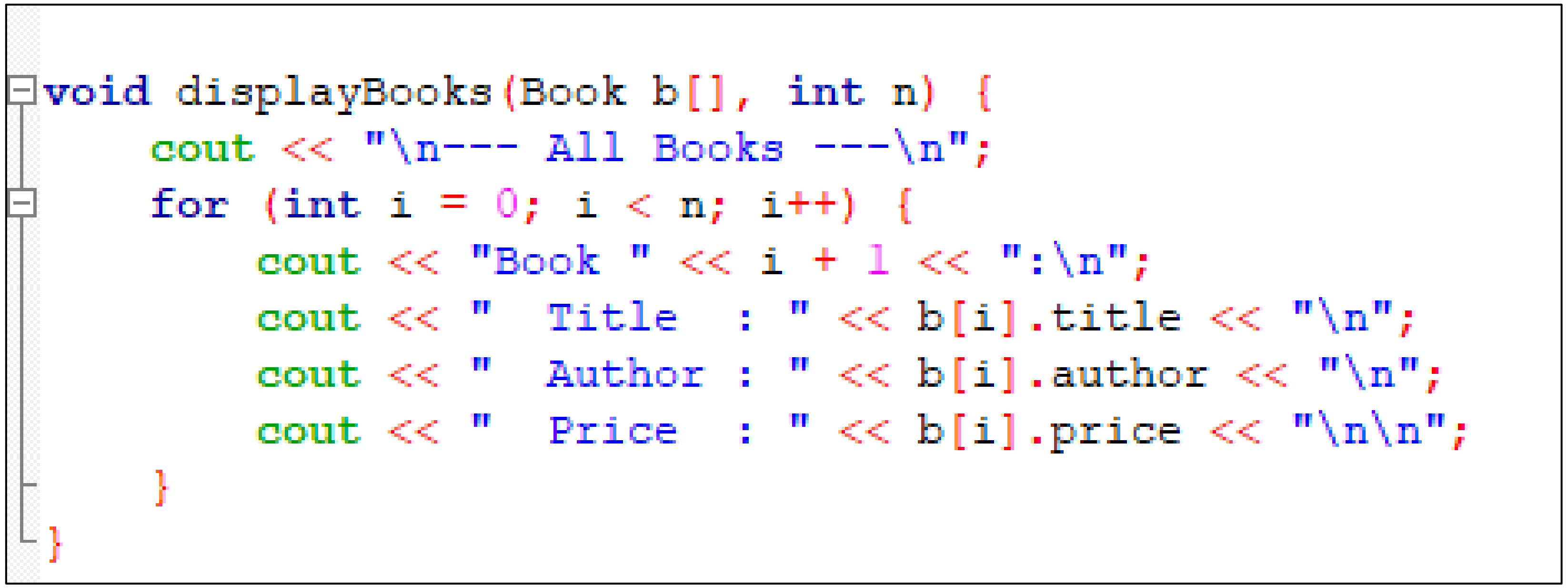
```
int main() {
    int N = 2;
    Book books[N];
    int choice;
    char key[100];
    inputBooks(books,N);

    do {
        cout << "\n1. Display all books";
        cout << "\n2. Search book by title";
        cout << "\n3. Search books by author";
        cout << "\n4. Exit";
        cout << "\nEnter choice: ";
        cin >> choice;
        switch (choice) {
        case 1:
            displayBooks(books, N);
            break;
        case 2:
            cout << "Enter title to search: ";
            cin>>key;
            searchBookByTitle(books, N, key);
            break;
        case 3:
            cout << "Enter author to search: ";
            cin>>key;
            searchBooksByAuthor(books, N, key);
            break;
        case 4:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice.\n";
        }
    } while (choice != 4);
    return 0;
}
```

Inputbooks function



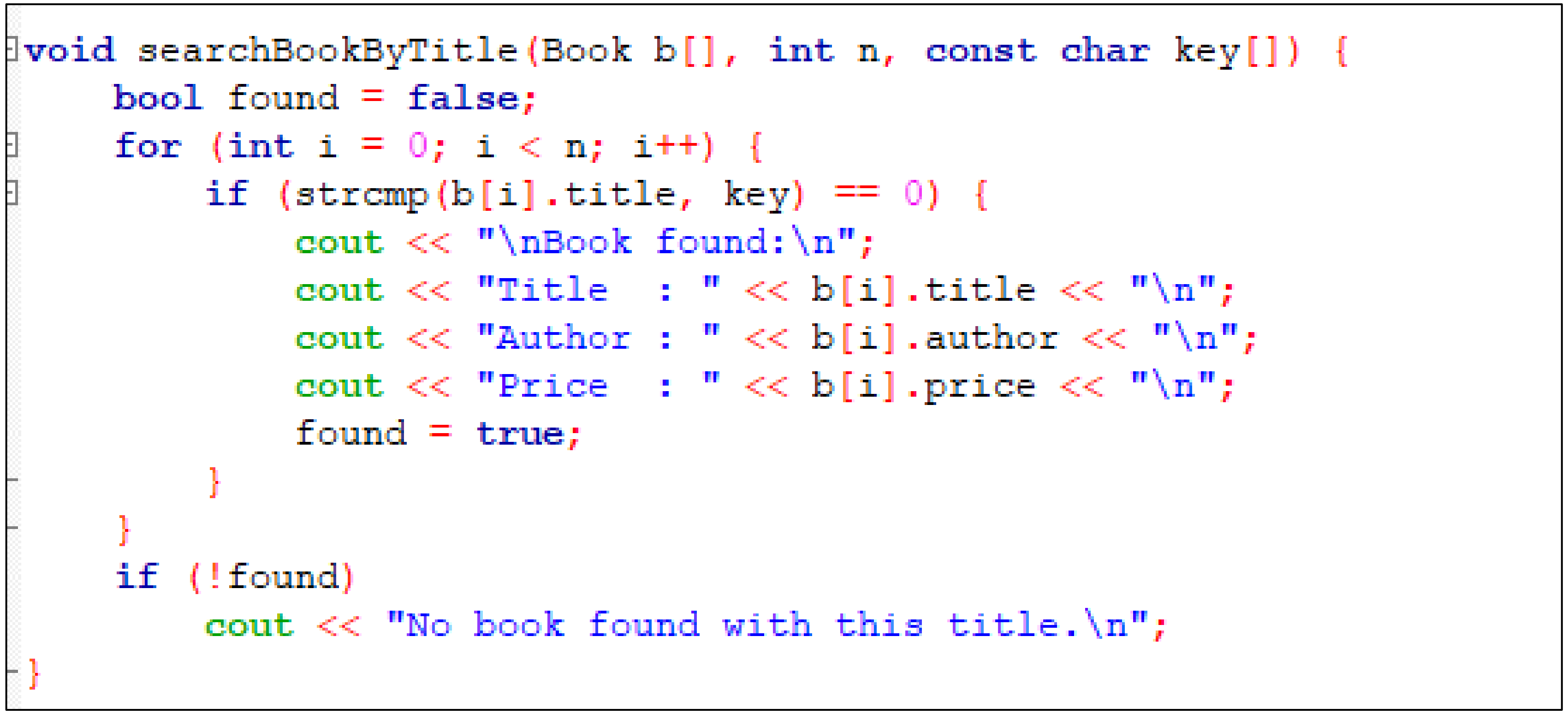
```
void inputBooks(Book b[], int n) {
    for (int i = 0; i < n; i++) {
        cout << "\nEnter details of Book " << i + 1 << ":\n";
        cout << "Title  : ";
        cin>>b[i].title;

        cout << "Author : ";
        cin>>b[i].author;

        cout << "Price  : ";
        cin >> b[i].price;
    }
}
```

Displaybooks function



```
void displayBooks(Book b[], int n) {
    cout << "\n--- All Books ---\n";
    for (int i = 0; i < n; i++) {
        cout << "Book " << i + 1 << ":\n";
        cout << "  Title  : " << b[i].title << "\n";
        cout << "  Author : " << b[i].author << "\n";
        cout << "  Price  : " << b[i].price << "\n\n";
    }
}
```

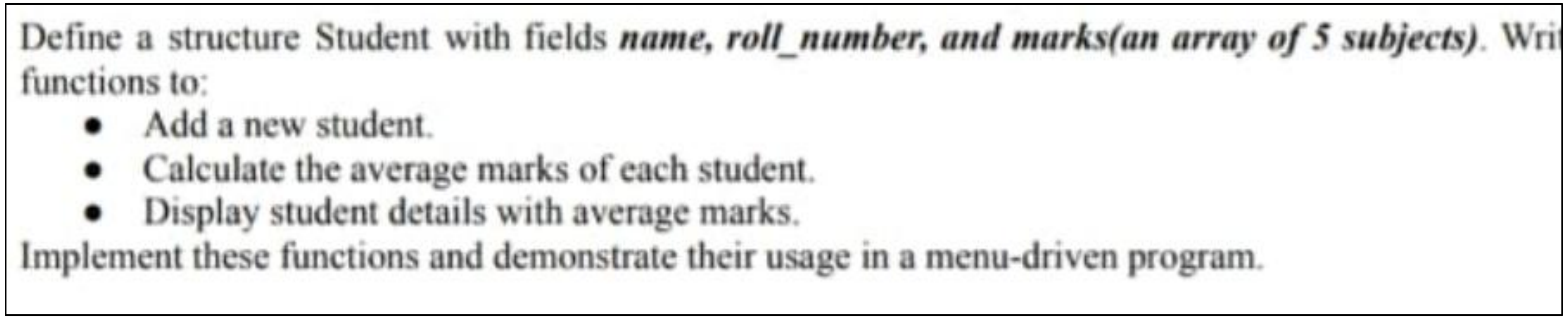
Searchbooks by title function



```
void searchBookByTitle(Book b[], int n, const char key[]) {
    bool found = false;
    for (int i = 0; i < n; i++) {
        if (strcmp(b[i].title, key) == 0) {
            cout << "\nBook found:\n";
            cout << "Title  : " << b[i].title << "\n";
            cout << "Author : " << b[i].author << "\n";
            cout << "Price  : " << b[i].price << "\n";
            found = true;
        }
    }
    if (!found)
        cout << "No book found with this title.\n";
}
```

Searchbooks by author function (try)

Programming question(try)

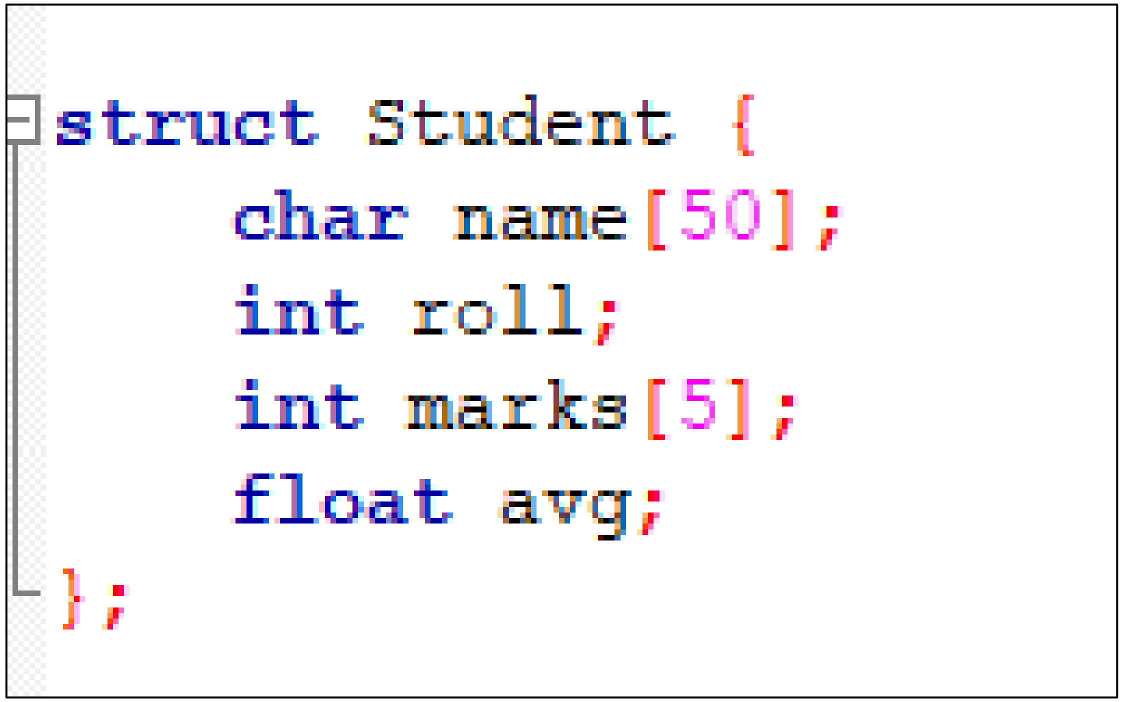


Define a structure Student with fields ***name, roll_number, and marks(an array of 5 subjects)***. Wri
functions to:

- Add a new student.
- Calculate the average marks of each student.
- Display student details with average marks.

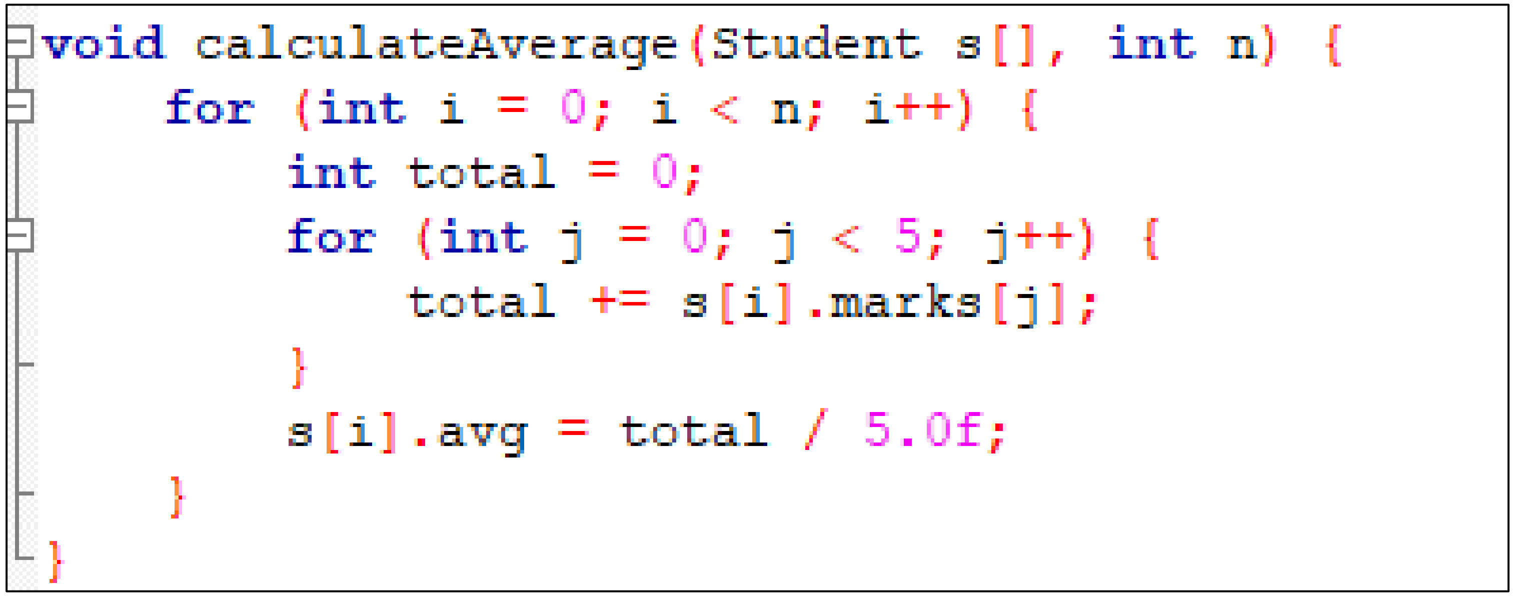
Implement these functions and demonstrate their usage in a menu-driven program.

Defining the structure



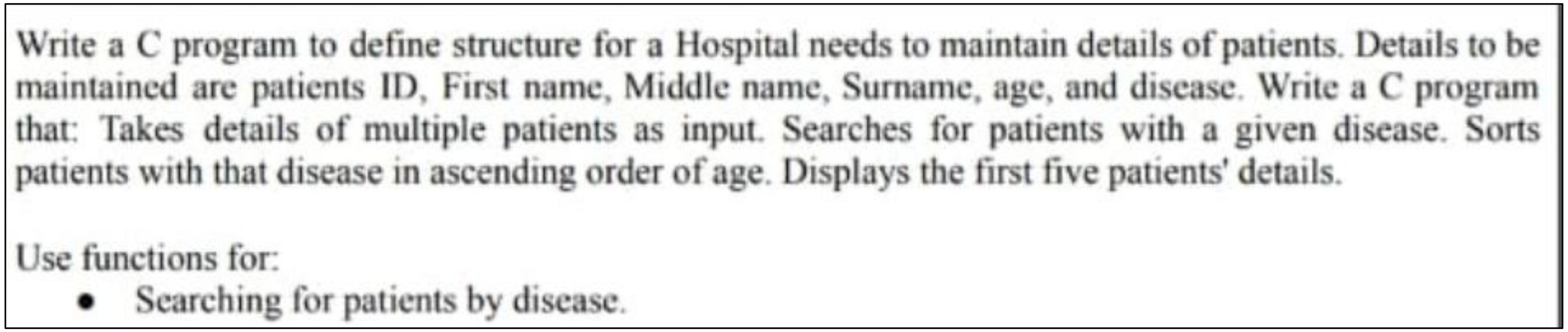
```c
struct Student {
        char name[50];
        int roll;
        int marks[5];
        float avg;
};
```

Function Calculate average marks of each student



```
void calculateAverage(Student s[], int n) {
    for (int i = 0; i < n; i++) {
        int total = 0;
        for (int j = 0; j < 5; j++) {
            total += s[i].marks[j];
        }
        s[i].avg = total / 5.0f;
    }
}
```

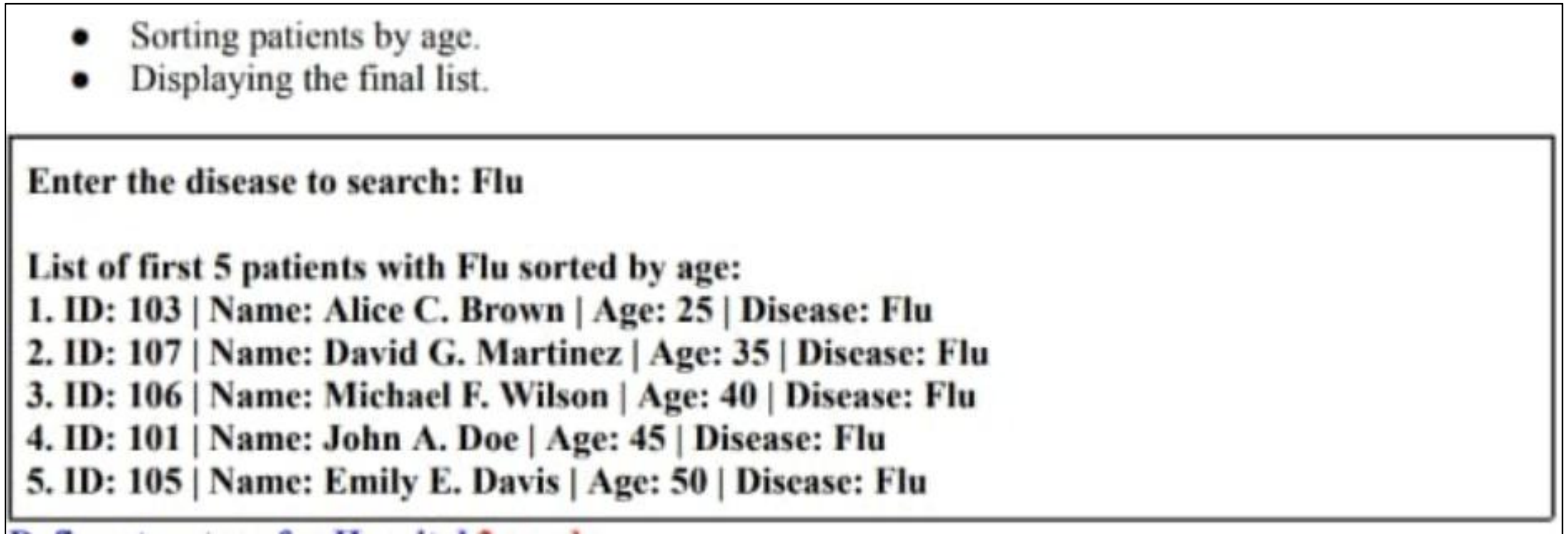
Programming question 3



Write a C program to define structure for a Hospital needs to maintain details of patients. Details to be maintained are patients ID, First name, Middle name, Surname, age, and disease. Write a C program that: Takes details of multiple patients as input. Searches for patients with a given disease. Sorts patients with that disease in ascending order of age. Displays the first five patients' details.

Use functions for:

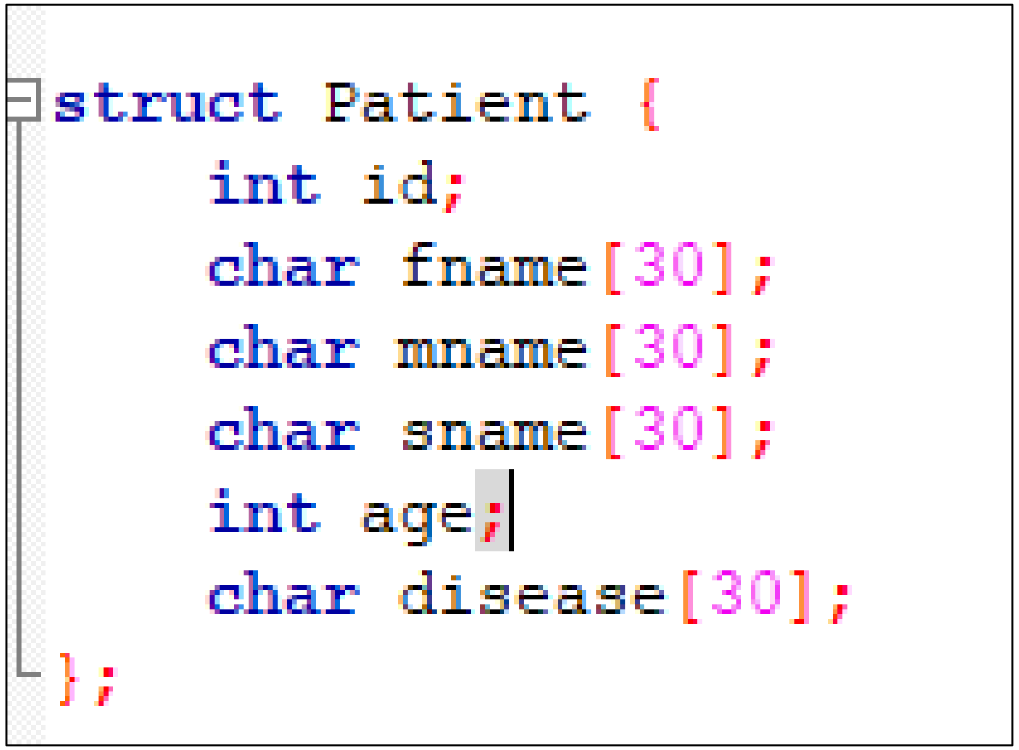
- Searching for patients by disease.



- Sorting patients by age.
- Displaying the final list.

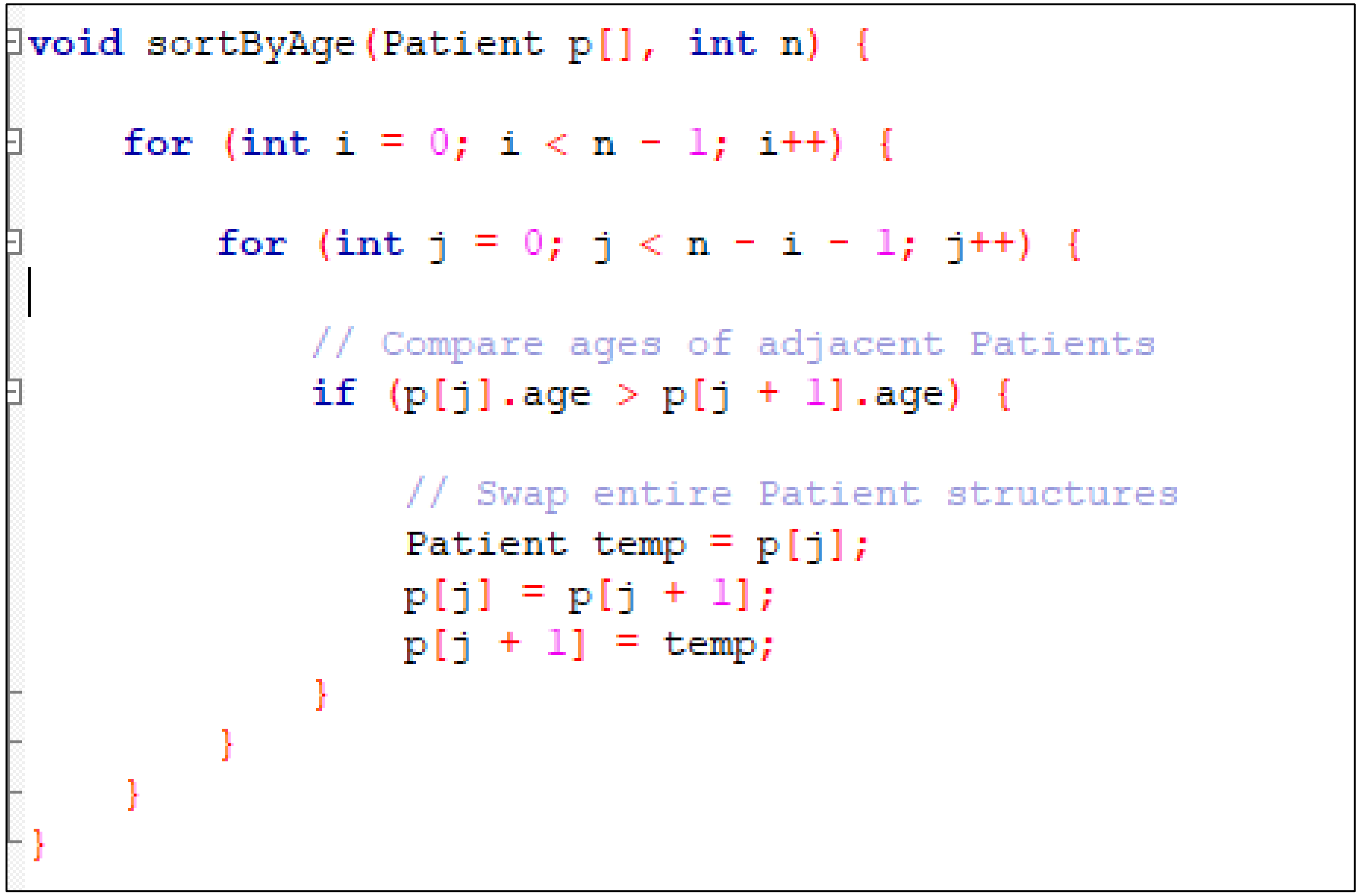
```
Enter the disease to search: Flu

List of first 5 patients with Flu sorted by age:
1. ID: 103 | Name: Alice C. Brown | Age: 25 | Disease: Flu
2. ID: 107 | Name: David G. Martinez | Age: 35 | Disease: Flu
3. ID: 106 | Name: Michael F. Wilson | Age: 40 | Disease: Flu
4. ID: 101 | Name: John A. Doe | Age: 45 | Disease: Flu
5. ID: 105 | Name: Emily E. Davis | Age: 50 | Disease: Flu
```

Defining Structure



```
struct Patient {
        int id;
        char fname[30];
        char mname[30];
        char sname[30];
        int age;
        char disease[30];
};
```

Sorting patients by age



```
void sortByAge(Patient p[], int n) {

    for (int i = 0; i < n - 1; i++) {

        for (int j = 0; j < n - i - 1; j++) {

            // Compare ages of adjacent Patients
            if (p[j].age > p[j + 1].age) {

                // Swap entire Patient structures
                Patient temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}
```