

C++ Data Structures

C/C++ arrays allow you to define variables that combine several data items of the same kind, but **structure** is another user defined data type which allows you to combine data items of different kinds.

Structures are used to represent a record, suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

- Title
- Author
- Subject
- Book ID

Defining a Structure

To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member, for your program. The format of the struct statement is this –

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

The **structure tag** is optional and each member definition is a normal variable definition, such as `int i;` or `float f;` or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure –

```
struct Books {  
    ... char title[50];  
    ... char author[50];  
    ... char subject[100];  
    ... int book_id;  
} book;
```

Accessing Structure Members

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use **struct** keyword to define variables of structure type. Following is the example to explain usage of structure –

Live Demo

```
#include <iostream>
#include <cstring>

using namespace std;

struct Books {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};

int main() {
    struct Books Book1;           // Declare Book1 of type Book
    struct Books Book2;           // Declare Book2 of type Book

    // book 1 specification
    strcpy( Book1.title, "Learn C++ Programming");
    strcpy( Book1.author, "Chand Miyan");
    strcpy( Book1.subject, "C++ Programming");
    Book1.book_id = 6495407;

    // book 2 specification
    strcpy( Book2.title, "Telecom Billing");
    strcpy( Book2.author, "Yakit Singha");
    strcpy( Book2.subject, "Telecom");
    Book2.book_id = 6495700;

    // Print Book1 info
    cout << "Book 1 title : " << Book1.title <<endl;
    cout << "Book 1 author : " << Book1.author <<endl;
    cout << "Book 1 subject : " << Book1.subject <<endl;
    cout << "Book 1 id : " << Book1.book_id <<endl;

    // Print Book2 info
    cout << "Book 2 title : " << Book2.title <<endl;
    cout << "Book 2 author : " << Book2.author <<endl;
    cout << "Book 2 subject : " << Book2.subject <<endl;
    cout << "Book 2 id : " << Book2.book_id <<endl;
```

```

    return 0;
}

```

When the above code is compiled and executed, it produces the following result –

```

Book 1 title : Learn C++ Programming
Book 1 author : Chand Miyan
Book 1 subject : C++ Programming
Book 1 id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Yakrit Singha
Book 2 subject : Telecom
Book 2 id : 6495700

```

Structures as Function Arguments

You can pass a structure as a function argument in very similar way as you pass any other variable or pointer. You would access structure variables in the similar way as you have accessed in the above example –

Live Demo

```

#include <iostream>
#include <cstring>

using namespace std;
void printBook( struct Books book );

struct Books {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};

int main() {
    struct Books Book1;           // Declare Book1 of type Book
    struct Books Book2;           // Declare Book2 of type Book

    // book 1 specification
    strcpy( Book1.title, "Learn C++ Programming");
    strcpy( Book1.author, "Chand Miyan");
    strcpy( Book1.subject, "C++ Programming");
    Book1.book_id = 6495407;
}

```

```

... // book 2 specification
strcpy( Book2.title, "Telecom Billing");
strcpy( Book2.author, "Yakit Singha");
strcpy( Book2.subject, "Telecom");
Book2.book_id = 6495700;

... // Print Book1 info
printBook( Book1 );

... // Print Book2 info
printBook( Book2 );

... return 0;
}

void printBook( struct Books book ) {
    cout << "Book title : " << book.title <<endl;
    cout << "Book author : " << book.author <<endl;
    cout << "Book subject : " << book.subject <<endl;
    cout << "Book id : " << book.book_id <<endl;
}

```

When the above code is compiled and executed, it produces the following result –

```

Book title : Learn C++ Programming
Book author : Chand Miyan
Book subject : C++ Programming
Book id : 6495407
Book title : Telecom Billing
Book author : Yakrit Singha
Book subject : Telecom
Book id : 6495700

```

Pointers to Structures

You can define pointers to structures in very similar way as you define pointer to any other variable as follows –

```
struct Books *struct_pointer;
```

Now, you can store the address of a structure variable in the above defined pointer variable. To find the address of a structure variable, place the & operator before the structure's name as follows –

```
struct_pointer = &Book1;
```

To access the members of a structure using a pointer to that structure, you must use the -> operator as follows –

```
struct_pointer->title;
```

Let us re-write above example using structure pointer, hope this will be easy for you to understand the concept –

Live Demo

```
#include <iostream>
#include <cstring>

...

using namespace std;
void printBook( struct Books *book );

struct Books {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};

int main() {
    struct Books Book1;           // Declare Book1 of type Book
    struct Books Book2;           // Declare Book2 of type Book
    ...

    // Book 1 specification
    strcpy( Book1.title, "Learn C++ Programming");
    strcpy( Book1.author, "Chand Miyan");
    strcpy( Book1.subject, "C++ Programming");
    Book1.book_id = 6495407;

    ...

    // Book 2 specification
    strcpy( Book2.title, "Telecom Billing");
    strcpy( Book2.author, "Yakit Singha");
    strcpy( Book2.subject, "Telecom");
    Book2.book_id = 6495700;

    ...

    // Print Book1 info, passing address of structure
    printBook( &Book1 );

    ...

    // Print Book2 info, passing address of structure
    printBook( &Book2 );
```

```

    return 0;
}

// This function accept pointer to structure as parameter.
void printBook( struct Books *book ) {
    cout << "Book title : " << book->title <<endl;
    cout << "Book author : " << book->author <<endl;
    cout << "Book subject : " << book->subject <<endl;
    cout << "Book id : " << book->book_id <<endl;
}

```

When the above code is compiled and executed, it produces the following result –

```

Book title : Learn C++ Programming
Book author : Chand Miyan
Book subject : C++ Programming
Book id : 6495407
Book title : Telecom Billing
Book author : Yakrit Singha
Book subject : Telecom
Book id : 6495700

```

The typedef Keyword

There is an easier way to define structs or you could "alias" types you create. For example –

```

typedef struct {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
} Books;

```

Now, you can use *Books* directly to define variables of *Books* type without using struct keyword. Following is the example –

```
Books Book1, Book2;
```

You can use **typedef** keyword for non-structs as well as follows –

```
typedef long int *pint32;
```

```
point32 x, y, z;
```

x, y and z are all pointers to long ints.