Procedural programming

Lecture 8

Dr.eng. Zaid Shhedi

Data structures

- Data Structure,s Syntax

```
struct type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
```

Data structures

- **Type_name** is a name for the structure type,
- **object_name** can be a set of valid identifiers for objects that have the type of this structure.
- Within braces {}, there is a list with the **data members,** each one is specified with a type and a valid identifier as its name.
- Example:

```
struct product {
      int weight;
      double price;
 } ;
 product apple;
 product banana, melon;
```

Data structures

- This declares a structure type, called product, and defines it having two members: weight and price, each of a different fundamental type. This declaration creates a new type (product), which is then used to declare three objects (variables) of this type: apple, banana, and melon. Note how once product is declared, it is used just like any other type.
- At the end of the struct definition, there is a semicolon (;),
- Object_names can be used to directly declare objects of the structure type.

Data structures

```
struct product {
	int weight;
	double price;
} apple, banana, melon;
```

- Where object_names are specified, the type name (product) becomes optional
- It is important to differentiate between what is the structure type name (product), and what is an object of this type (apple, banana, and melon).

Data structures

- Once the three objects of a determined structure type(Product) are declared (apple, banana, and melon) its members can be accessed directly.
- The syntax for that is simply to insert a dot (.) between the object name and the member name. For example, we could operate with any of these elements as if they were standard **variables** of their respective types:

```
apple.weight
apple.price
banana.weight
banana.price
melon.weight
melon.price
```

Data structures

- Each one of these has the data type corresponding to the member they refer to: apple.weight, banana.weight, and melon.weight are of type int, while apple.price, banana.price, and melon.price are of type double.

Data structures

```
struct movies_t {
    string title;
    int year;
  } mine, yours;
```

Data structures

- The example shows how the members of an object act just as regular variables. For example, the member yours.year is a valid variable of type int, and mine.title is a valid variable of type string.
- But the objects mine and yours are also variables with a type (of type movies_t).
- Therefore, one of the features of data structures is the ability to refer to both their members individually or to the entire structure as a whole. In both cases using the same identifier: the name of the structure.

Data structures

```
#include <iostream>
#include <cstring>
using namespace std;
struct Books
{
char title[50];
char author[50];
char subject[100];
int book_id;
};
```

Data structures

```
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");
```

Data structures

```
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;
```

Data structures

```
cout << "Book 2 subject : " << Book2.subject <<endl;
cout << "Book 2 id : " << Book2.book_id <<endl;
return 0;
}
```

Output:

```
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 : Yakit Singha
Book 2 subject : Telecom
Book 2 id : 6495700
```

Structures as Function Arguments

```
#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;
};
```

Structures as Function Arguments

```
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;
```

Structures as Function Arguments

```
// 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;
}
```

Structures as Function Arguments

```
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;
}
```

Structures as Function Arguments

Output:

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

Pointers to structures

- Like any other type, structures can be pointed to by its own type of pointers:

```
struct movies_t {
    string title;
    int year;
};
movies_t  amovie;
movies_t *  pmovie;
```

Pointers to structures

- Here amovie is an object of structure type movies_t, and pmovie is a pointer to point to objects of structure type movies_t. Therefore, the following code would also be valid:

```
pmovie = &amovie;
```

Pointers to structures

- **Example:**

```
#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;
};
```

Pointers to structures

```
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;
```

Pointers to structures

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

Pointers to structures

Output:

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

Pointers to structures

- The arrow operator (->) is a dereference operator that is used exclusively with pointers to objects that have members. This operator serves to access the member of an object directly from its address. For example, in the example above:
- pmovie->title Is equivalent to (*pmovie).title

Expression	What is evaluated	Equivalent
a.b	Member b of object a	
a->b	Member b of object pointed to by a	(*a).b
*a.b	Value pointed to by member b of object a	*(a.b)

Nesting structures

- Structures can also be nested in such a way that an element of a structure is itself another structure:

```
struct movies_t {
  string title;
  int year;
};
struct friends_t {
  string name;
  string email;
  movies_t  favorite_movie;
} charlie, maria;
friends_t *  pfriends = &charlie;
```

Nesting structures

- All of the following expressions would be valid:

- charlie.name
- maria.favorite_movie.title
- charlie.favorite_movie.year
- pfriends->favorite_movie.year

- The last two expressions refer to the same member.