Lecture 7 Structures

Objectives

At the end of this lecture, you should be able to

- Explain the difference between simple data types and structures
- Use structures to represent data and operations with a structure
- Use arrays of structures
- Pass structures to a function
- Return a structure from a function

Read and Watch

- Swedish textbook 8.1, or
- English textbook chapter 6.1-4, or
- For beginners, it would be easier to read tutorialspoint.com
  - C Structures
- Video clip C Programming Tutorials https://www.youtube.com/playlist?list=PL6gx4Cwl9DGAKIXv8Yr6nhGJ9Vlcjyymq
  - 49

Simple data types

- Simple data types
  - integer types
  - floating types
  - char
- Only one value of the type can be stored in a variable of simple data types
- Array allows multiple values of **the same type** be stored in an array variable

Motivation for using structures

- in application handling books, we may use following variables to represent info about a book:
  - char title[50]; // title of the book
  - char author[50]; // who is the author
  - int pages; // number of pages
  - float price; // the cost of the book
  - float weight;
- It would be easier, if we could organize these related data into a single unit, or a variable can store this collection of related data (of different types).
  - It cannot be an array since data types are different
- The solution in C is called structures.

Array vs Structure

- An array is a collection of one or more variables (array elements) of the same type, grouped together under a single name for convenient handling
- A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling
- *If you have learnt some object-oriented language, you can think that a structure is a class without methods*

Example– book

- struct declaration
  - We define a structure to keep book info
  - user defined type

```
struct book {
    char title[50], author[50];
    int pages;
    float price;
    float weight;
};
```

- struct variable definition

```
struct book b1, b2; // defined 2 variables for 2 books.
```

Variable of a structure

- A struct declaration define a type and the list of members.
- Structure variables

```
struct book b1, b2;
```

defines variables b1 and b2, each of them is a structure of type struct book

– Memory allocation is done for the members

Access to a structure member

- Dot operator is used to access the structure members
  - structure-var-name.member
- You use the structure members in the same way as you use other variables
- Example

b1.pages = 276;

➔ it means the member ”pages” of variable b1 is assigned with a value of 276.

printf("Book title %s och costs %f kronor\n", b1.title, b1.price);

Example—accessing struct members using dot operator

- src file: book.c
- Read in book info – each member is used as a variable

```
struct book b1;

printf("Enter the book title: ");
fgets(b1.title, 50, stdin);  // question: why not using scanf( )?
printf("Enter the author: ");
fgets(b1.author, 50, stdin);
printf("Enter the number of pages: ");
scanf("%d", & b1.pages);
printf("Enter the price:");
scanf("%f", & b1.price);
scanf("Enter the weight of the book: ");
scanf("%f", & b1.weight);
```

structure operations

- Dot operator is used to access the structure members
- **Assignment** is allowed on structure variables

  struct book b1, b2;

  b2 = b1; // ok, b2 has the same value as b1
- **Comparison** is not allowed

  b2 == b1   // it is not ok!

  ➔ You need to implement comparison yourself
- Compare structure with array operations
  - Dot operator to access structure member b1.price
  - indexing to access array elements s[i]
  - Assignment is allowed for structures
  - assignment (=) is not allowed for arrays
  - comparison (==) does not work for both of them

Pass a structure to a function

- A structure can be passed to a function
- the **value is copied**
  - pass by value in C
- Example book.c
  - void printBookInfo (struct book b);

Return value from a function

- As we discussed, a function can only return one value from a function
- If we want to have multiple values returned?
  - put them in a structure
  - the function can return the whole structure as a single value

Return value from a function

- Example book.c

  struct book readInBook(void);

  to read in info about a book, and return the book as a structure

Problems with a large structure to/from a function

- Passing a structure to a function
  - the structure value is copied the argument,
- Returning a structure from a function
  - The value is copied to the variable receiving the returning value
- In both cases, it takes time for copying operating, if the structure is large
  - In the example, book.c:
  - On my computer, it takes 112 bytes to represent a book. So to pass a structure, it is required to copy 112 bytes
- Solution is to pass the address of the structure (or structure pointer)

Example—pointer to a structure

- To be more efficient in running time, ➔ use pointers to structures
- Pointer operator

  struct book b1;

  struct book *bptr; //bptr is a pointer to a structure

  bptr = & b1; //bptr points to b1
- using pointer to access the members
  - (*bptr).pages, (*bptr).price or
  - bptr->pages, bptr->price (-> is called the pointer operator)
- Example book.c
  - void printBookInfo2(struct book *bptr);

Arrays of structures

- Array of integers
  - **int** grades[40];
  - memory space is allocated to store 40 integers
- Array of struct bo0k
  - **struct book** b[100]; // store 100 books
  - memory space is allocated to store 100 structures
- sizeof ( )
  - Used to compute the size of any object at the compile-time
  - sizeof (int) ➔ gives the number of bytes for an int variable
  - sizeof (struct book) ➔ give the number of bytes for a struct bok variable
  - sizeof (grades) ➔ no. of bytes for array grades

typedef

- Creates new data type names
- Make programs more readable
- Examples
  - **typedef int** length; ➔ **length** len, maxlen;
  - **typedef char *** String; ➔ **String** p;