**Labsheet 10**

**Structures**

1. Define a structure called **Book**  to represent a bibliography entry. The structure should contain the following fields.

|  |  |
| --- | --- |
| **Field** | **Type** |
| Title | Character array |
| Author | Character array |
| Pages | Integer |
| Date Published | Integer |

Declare a variable of your book type and assign values to the fields. You will need to use **strcpy()** for the author and title. Use **printf()** to print out the values assigned.

1. Try assigning values to the structure variable when it is declared using structure initialization. Use **printf()** to print out the values you assigned.
2. Instead of assigning values explicitly, use scanf() to read values from the user directly into your structure variable and **printf()**  to print out the values you read in.
3. Write a function which reads in a bibliography entry. The function should take no arguments and return a book structure.

Inside the function declare a local book variable and read values into it from the user. Return this local variable as the return value of the function. Remember that the local variable goes away when the function ends, but the values will be copied as the return value. The prototype to the input function should look like this:

**struct Book input\_book( void );**

Write a main programto test your book input function. Don’t forget to assign the return value of the function to a book variable like this:

**struct Book my\_book;**

**my\_book = input\_book();**

1. Declare an array of **3 struct Book**  structures. Fill in the array using three calls to the book input function. Print out the values in each book structure of the array.
2. Write a function which takes two arguments: an array of book structures and the length of the array. The function should search through the array of books and return the index of the oldest book.
3. Define a structure to store a two dimensional point. The structure needs only two fields, the **x** and **y** coordinates of the point. The point structure should look like this:

**Struct Point {**

**int x, y;**

**};**

Write a function which takes two point structures as parameters and returns the distance between the two points. The prototype for the distance function should look like this:

**Double dist( struct Point p1, struct Point p2);**

To compute the distance between the two points, use the formula:

distance =

To compute the square roo, you can use the library function **sqrt().** The prototype for the **sqrt()** is:

**double sqrt(double);**

1. Write a function **swap()** which takes two pointers to the following structure:

**Struct Point {**

**int x, y;**

**};**

and returns **void**. The function should swap the values stored in the two points. Write a main program and test your function.

1. Define two structures: One to represent a two dimensional point and one to represent a line. The point structure needs two fields: two float variables to represent the x and y coordinates of the point. The line structure also needs two fields: two point structures to store two points on the line.

Write a function which takes a line structure as an argument and returns the slope of the line. Use the following formula to compute the slope:

Slope =

The prototype for the slope function should look like this:

**float slope( struct Line l );**

Write a main program which tests the slope function.

**Pointers to Structures**

1. Cash Register

Implement a very simple cash register using a structure to store the data. Write several functions which operate on the cash register structure. Each function will take a pointer to a cash register structure as one of its parameters.

The structure and some sample function prototypes are shown below. The **init()** function should load the specified bills into the cash register. The **add\_notes()** and **remove\_notes()** functions add or remove the given numbers of notes. **value()** computes the total amount stored, and **inventory()**  function prints out the number of each type of bill and the total value. Have the **inventory()** function make use of **value()** instead of doing the calculation itself.

**typedef struct CashRegister {**

**int tens;**

**int fives;**

**int ones;**

**} CashRegister;**

**void Init(CashRegister \*cr, int tens, int fives, int ones);**

**void add\_notes(CashRegister \*cr, int tens, int fives, int ones);**

**void remove\_notes(CashRegister \*cr, int tens, int fives, int ones);**

**int value(CashRegister \*cr);**

**void print(CashRegister \*cr);**

**\*\*\*\*\*\*\*\*\*\***