CPT109: C Programming & Software Engineering I

Lecture 9: Data Structures & Sorting & pbook

Soon Phei Tin
Office: SD531
Email: soon.tin@xjtlu.edu.cn

Group Project

- Report
  - Report should be well presented: -
    - Structured and formatted professionally
    - Cover page
    - Page header and footer
    - Written in a concise manner
    - Figures with captions
    - Figures clear and useful
  - Use of English
    - Spelling mistakes
    - Grammar

Group Project

- Specification
  - Should be a clear statement about what the program is for and what it can do.
    - General
    - What the program can/can’t do

Group Project

- Analysis
  - To fulfil the specification, what modules/functions do you have and how they interact with which other?
  - Inputs and outputs of each module?
  - What data structures are being used?
  - Process? If the process is complex, model it using diagrams.

Group Project

- Implementation
  - Self-documented code
  - Avoid use of goto and label
  - Write an easy to understand code
  - Appropriate use of variable and loops
  - Use local variable (unless good rationale for global)
  - Sensible choice of data structures and their application

Group Project

- Robustness
  - Testing for successful operation such as malloc, fopen, fclose.
  - Release all resources after use such as free(), fclose().
  - Testing/handling of user input.
  - Careful algorithm design with plenty of testing.

Group Project

- Testing
  - Describe your testing strategy.
  - Setting up test plan for integration and system testing.
  - Description for the fail test.
  - Description of any changes made as a result of testing
  - Describe and explain any bugs found and not fixed.

Outline of Today's Lecture (9)

- Data structures:
  - Arrays and Linked Lists
  - Binary Trees
  - Hash Tables
- Sorting
  - Bubble sort
  - Selection sort
  - Merge sort
- Phone Book programs (array, element, linked list)

Data Structures

Searching

- Much of what we do with a computer involves sorting and searching data e.g. internet searches
- We save data in order to be able to retrieve it later.
- The time it takes to find an item of stored data is strongly linked to the way in which it is stored.
- Data should be stored to minimise the time it takes to interact with it. Where interaction could be: viewing, editing, deleting, inserting, sorting.

Data Structures

- We have so far considered two basic data structures.
  - Arrays – sequentially ordered data elements of predefined size.
  - Linked List – dynamically created and distributed data elements linked by the data elements themselves.
- We have also considered the advantages and disadvantages of the two, but let’s recap a little…

Data Structures – Arrays

- The array is a basic data structure, consisting of a group of elements all with the same data type, stored in a sequential block of memory
- Advantages
  - Easy to access by index or pointer (incrementing or decrementing either moves through the elements)
  - Easy to perform a search
  - Fixed memory size (often known at compile time)
- Disadvantages
  - Fixed memory size (too little or too much)
  - Sorting is time consuming (whole elements must be copied)
  - Inserting and deleting is also awkward

Data Structures – Linked List

- The linked list is constructed of structure variables, each dynamically created and linked by including a pointer to the next element in the list within the structure.
- Advantages
  - Length of the list is only limited by computer memory
  - Sorting is accomplished by re-ordering the pointers
  - Deletion/insertion accomplished the same way
- Disadvantages
  - Searching is not simple like the array
  - Searching is in one direction (unless doubly linked)

Array or Linked List?

- If a situation calls for a list that is continuously resized with frequent insertions/deletions but that isn't searched often, linked lists are a better choice.
- If a situation calls for a stable list with only occasional insertions/deletions but that are searched often, an array is the better choice.

Other Data Structures

- There are a number of other standard structures:
  - Tree (binary tree)
  - Stack
  - Queue
  - Hash Table
  - There are more…
- These structures are built on one or both of the properties of the linked list or array.

Binary Search Tree

- A binary tree is a data structure that like a linked list is built on the structure data type.
- new elements can be created dynamically
- Each element points to two other elements based on some quantity relationship of the data.
- The order of elements is also based on some quantity relationship of the data.

Binary Search Tree - Example

Say we wanted to store all of the words in a book in a list ordered by the number of letters in each word. Let’s consider the passage “I love being an electronic design engineer”

I	1	electronic	10
love	4	design	6
being	5	engineer	8
an	2		

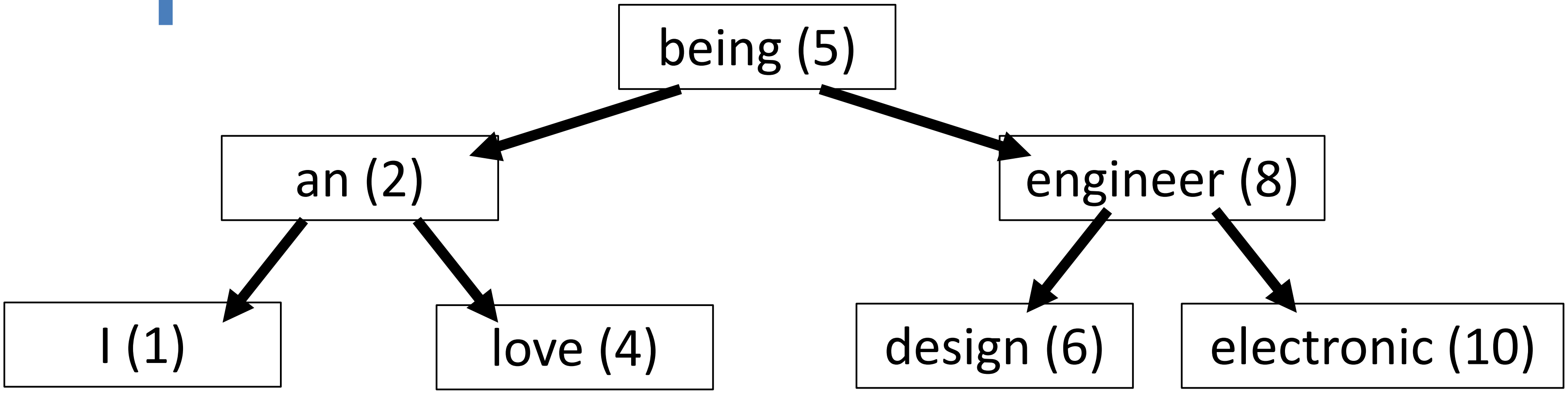
If stored in numerical order in a linked list, to find the word with the largest number of letters (electronic) we would have to traverse the whole list (7 steps).

Binary Search Tree - Example

I	1	electronic	10
love	4	design	6
being	5	engineer	8
an	2		

- Each data element is stored as a node
- Each node has a left and right branch (pointers)
  - left <=
  - right >
- The head node is the centre value of the set (5)

Binary Search Tree - Example



being (5)

an (2)

engineer (8)

I (1)

love (4)

design (6)

electronic (10)

- Balanced binary tree, designed for fast searching since it is sorted. Max 2 steps.
- All data is read into the program then sorted.
- Insertion/deletion – requires re-sort

```
struct node{    char *word;
                int count;
                struct node *left;
                struct node *right;};
```

Lookup Table (Hash Table)

- What if you could write an expression which based on a data input would return a unique but ordered integer (hash value).
- Creating an array and placing the data at that integer element number would make sorting and searching unnecessary.
- Simply calculate the hash value and go direct to the correct element.

Lookup Table (Hash Table)

Example:

- Data is contact list of phone numbers and family names.
- Want to be able to find the number by name.

Hash design:

- Create an array with 26 elements.
- Calculate hash value as ascii value of first letter of name – 101 (this would make 'A' – 101 = 0).
- Place name and number at element calculated.

Lookup Table (Hash Table)

Problems:

- Only enough elements in array to save one name and number for each letter
- Leads to potential collisions

Solutions:

- Make array larger e.g. 10 elements per letter.
- hash value = (‘char value’-101) * 10
- For placing or searching a name, each name with the same hash value stored in consecutive elements

Lookup Table (Hash Table)

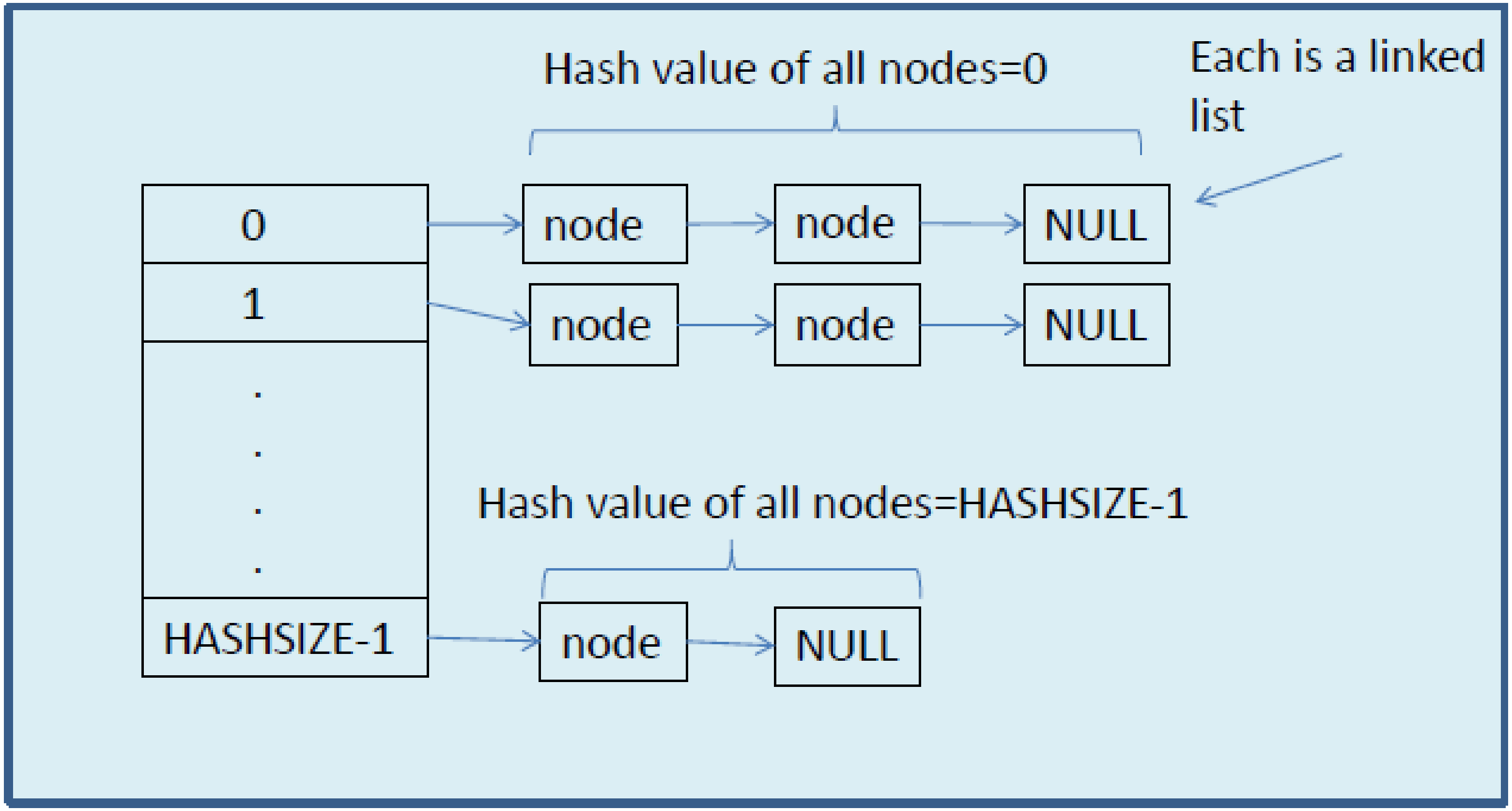
Problems:

- Still limited in size and will lead to collisions eventually.

Solutions:

- At each array element create a linked list to hold all data for names that produce the same hash value.

Lookup Table (Hash Table)



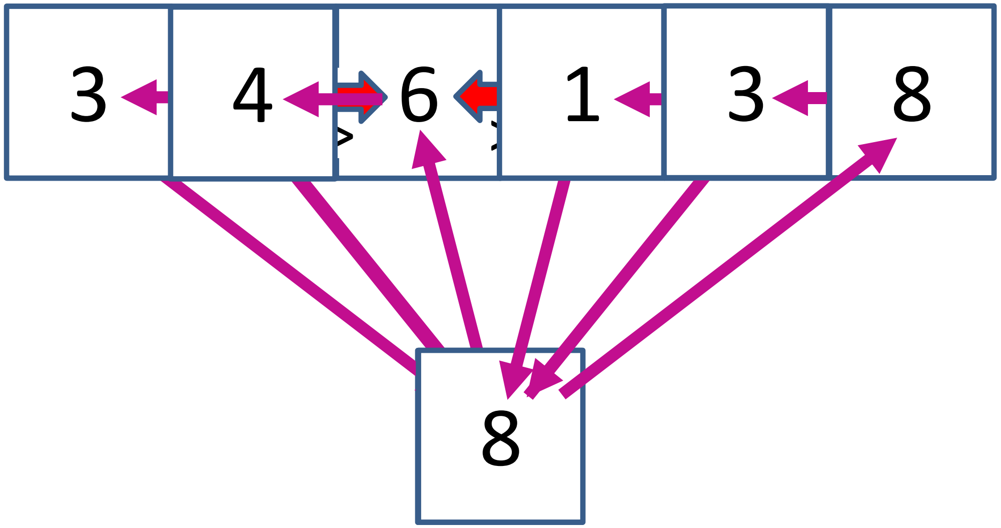
Sorting

Sorting

- There are many methods of sorting data stored in arrays:
    - Bubble sort
    - Selection sort
    - Merge sort
    - Quick sort
    - Insertion sort
    - there are more…

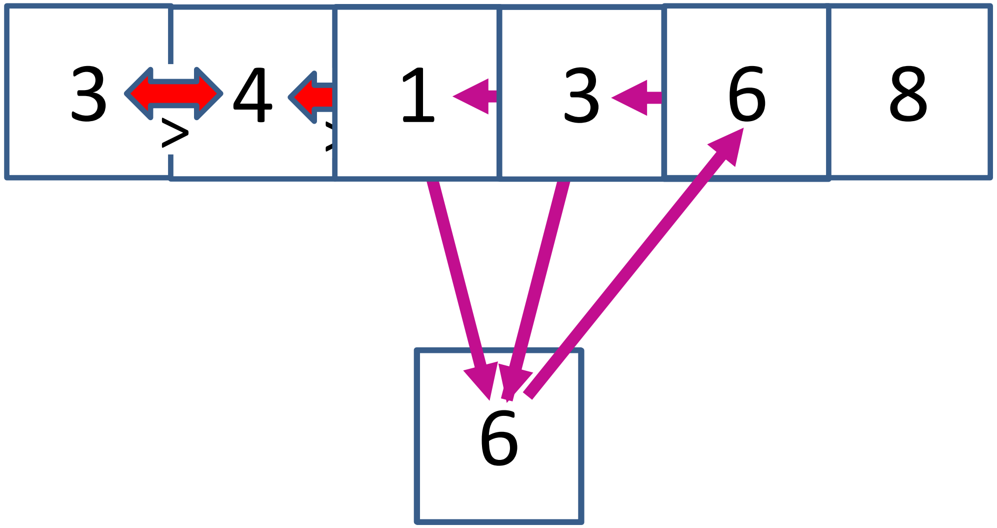
Bubble Sort

1st Pass



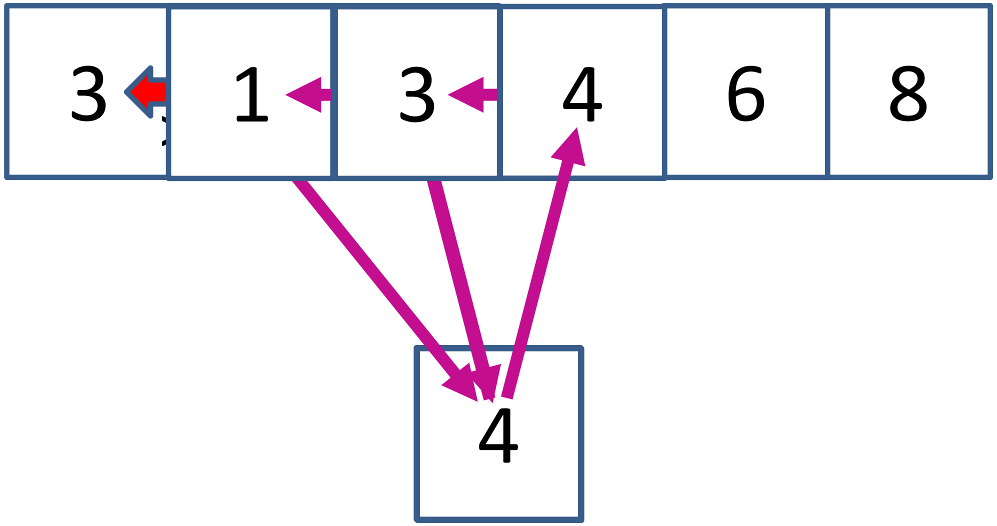
Bubble Sort

2nd Pass



Bubble Sort

3rd Pass

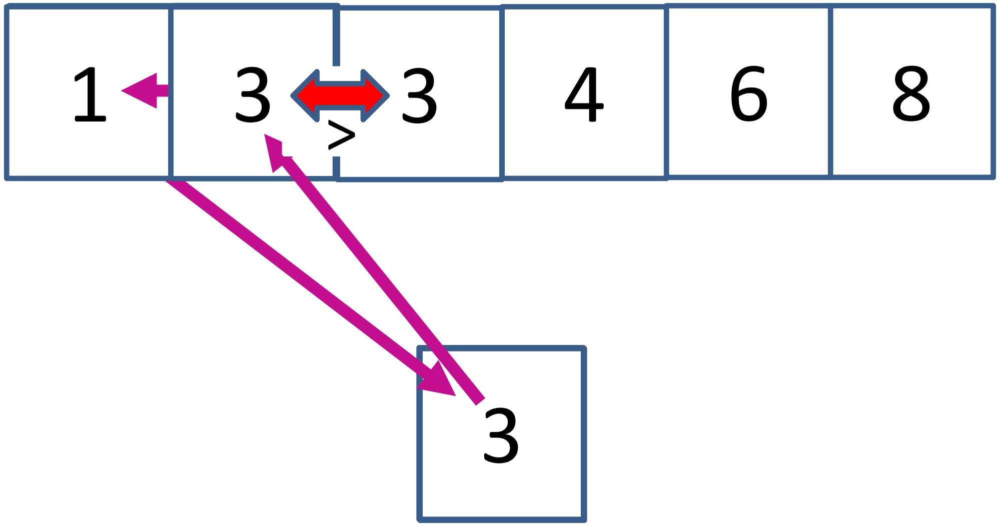


3 1 3 4 6 8

4

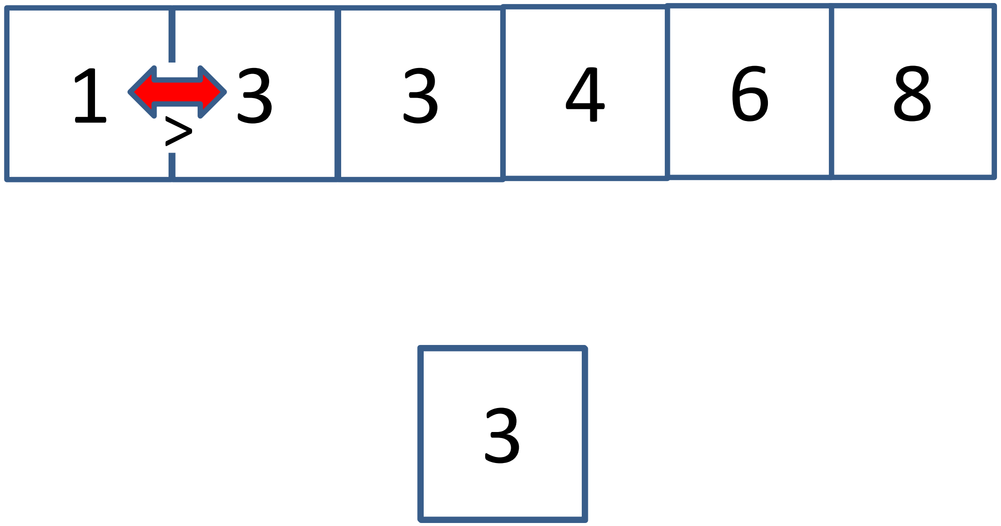
Bubble Sort

4th Pass



Bubble Sort

5th Pass

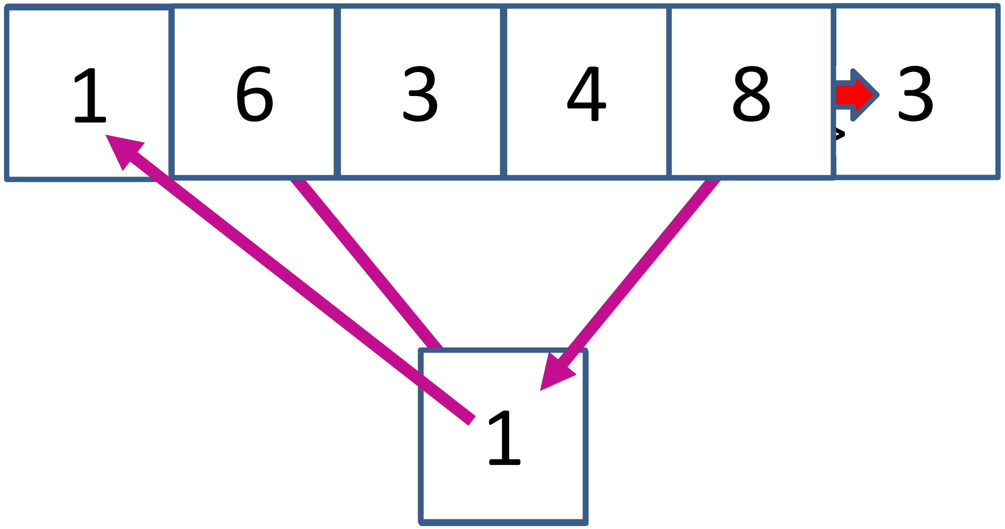


1	3	3	4	6	8

3

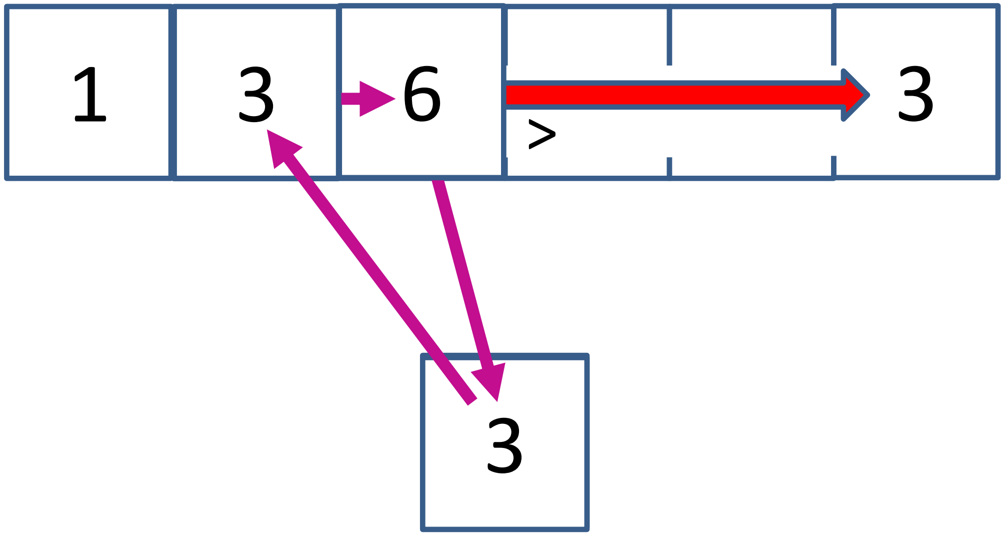
Selection Sort

1st Pass



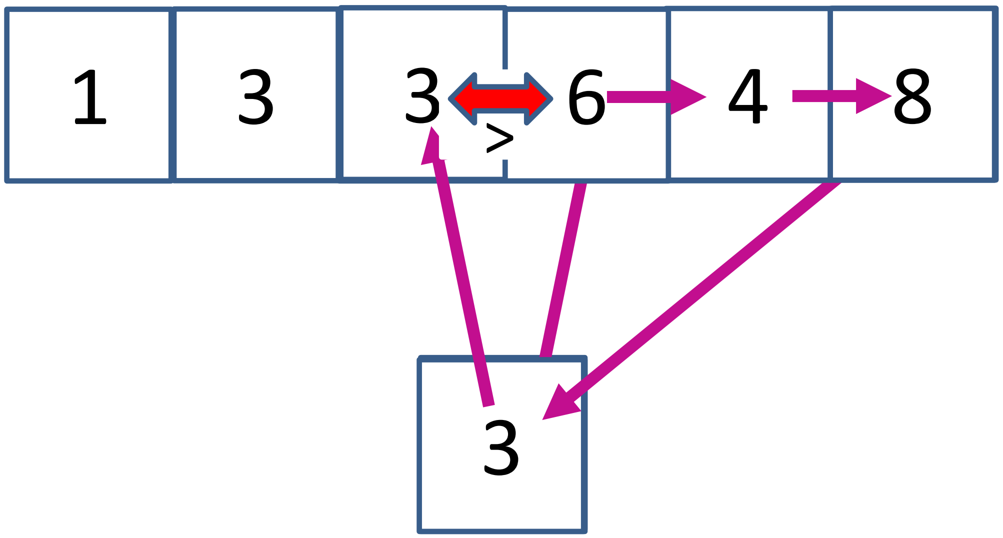
Selection Sort

2nd Pass



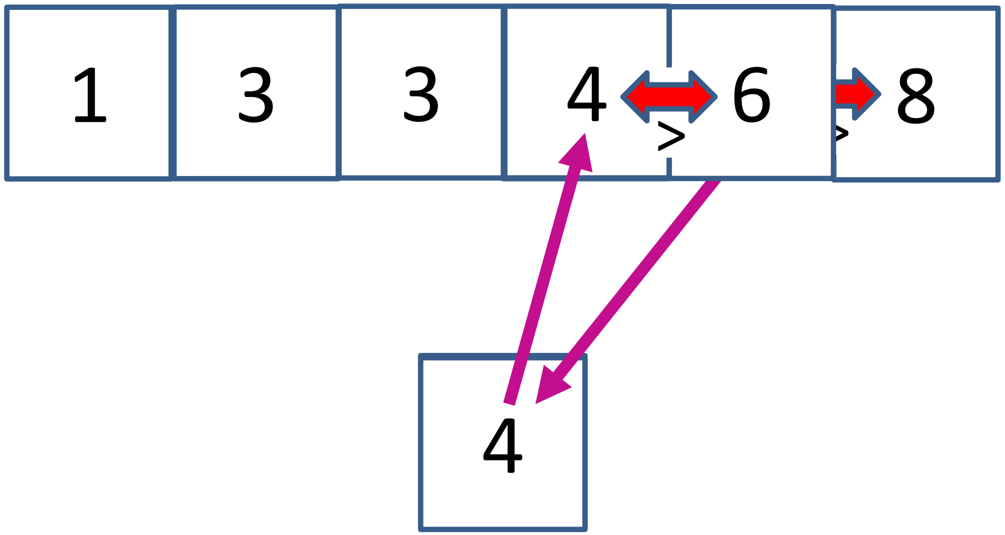
Selection Sort

3rd Pass



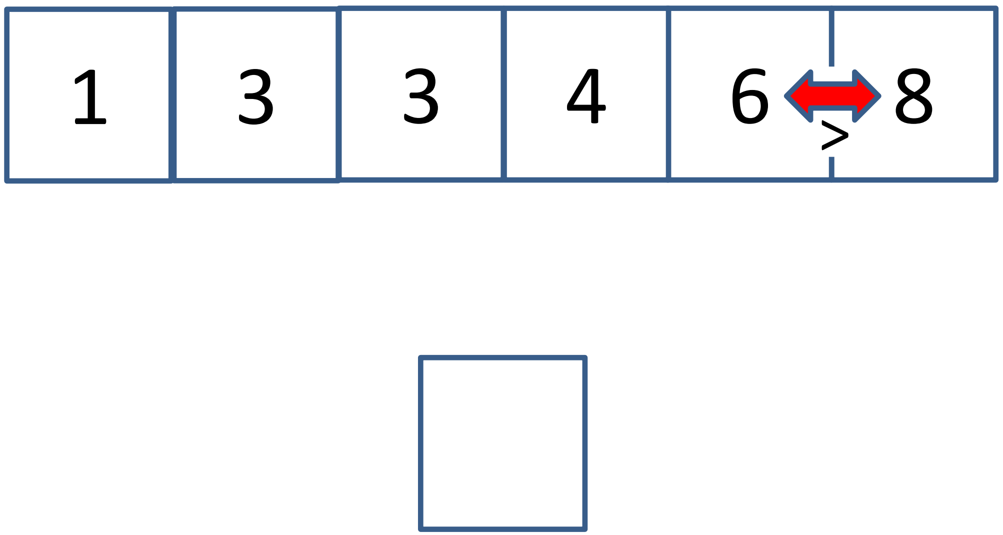
Selection Sort

4th Pass



Selection Sort

5th Pass

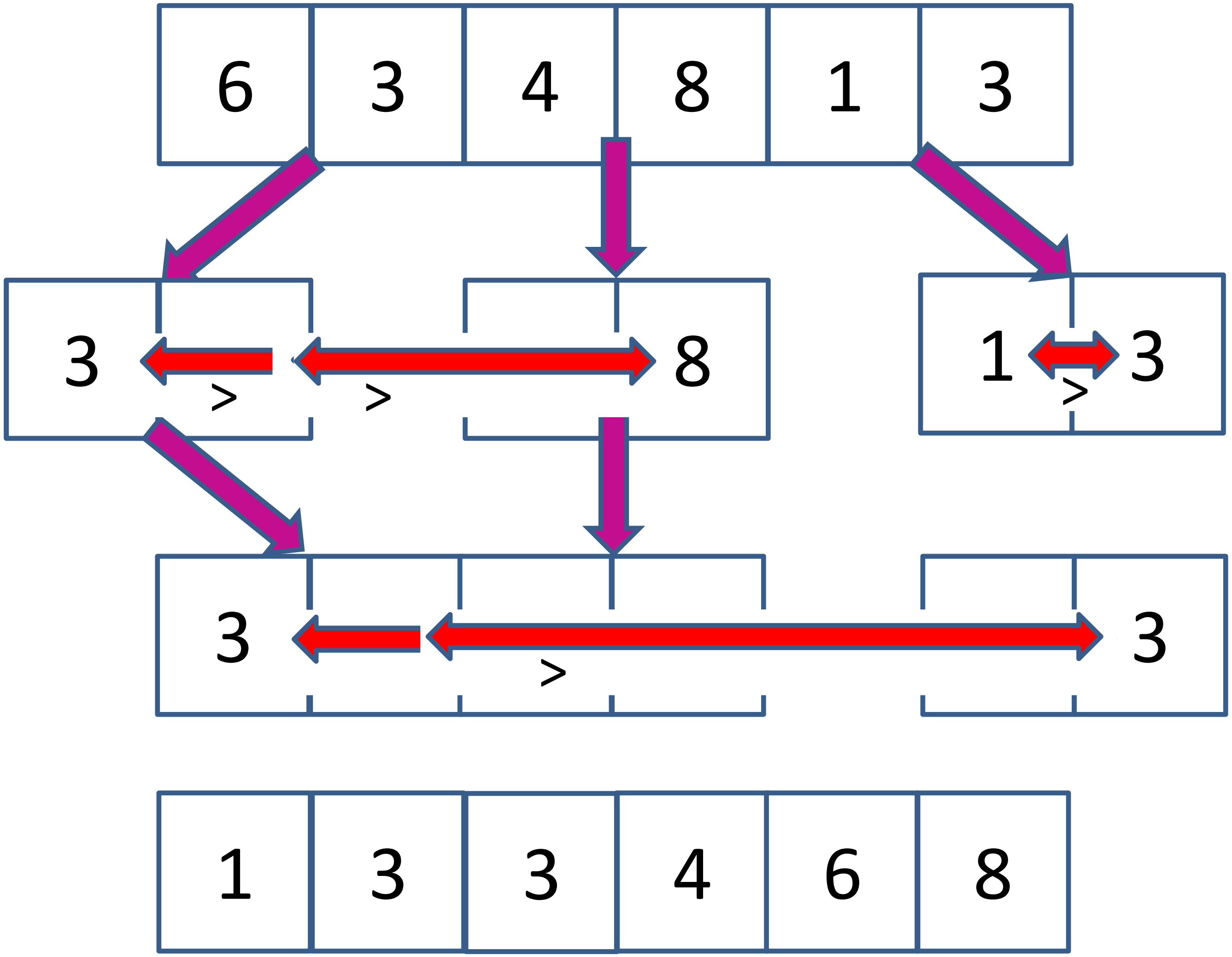


1	3	3	4	6	8

>

Merge Sort

Divide and conquer



Phone Book…

Phonebook program

Using array or linked list

A program for contact information.

- Contacts can be added.
- Contacts can be deleted.
- Contacts can be viewed by entering the name.
- Contacts are stored in a file.
- Different files can be used.

Copying whole file to array

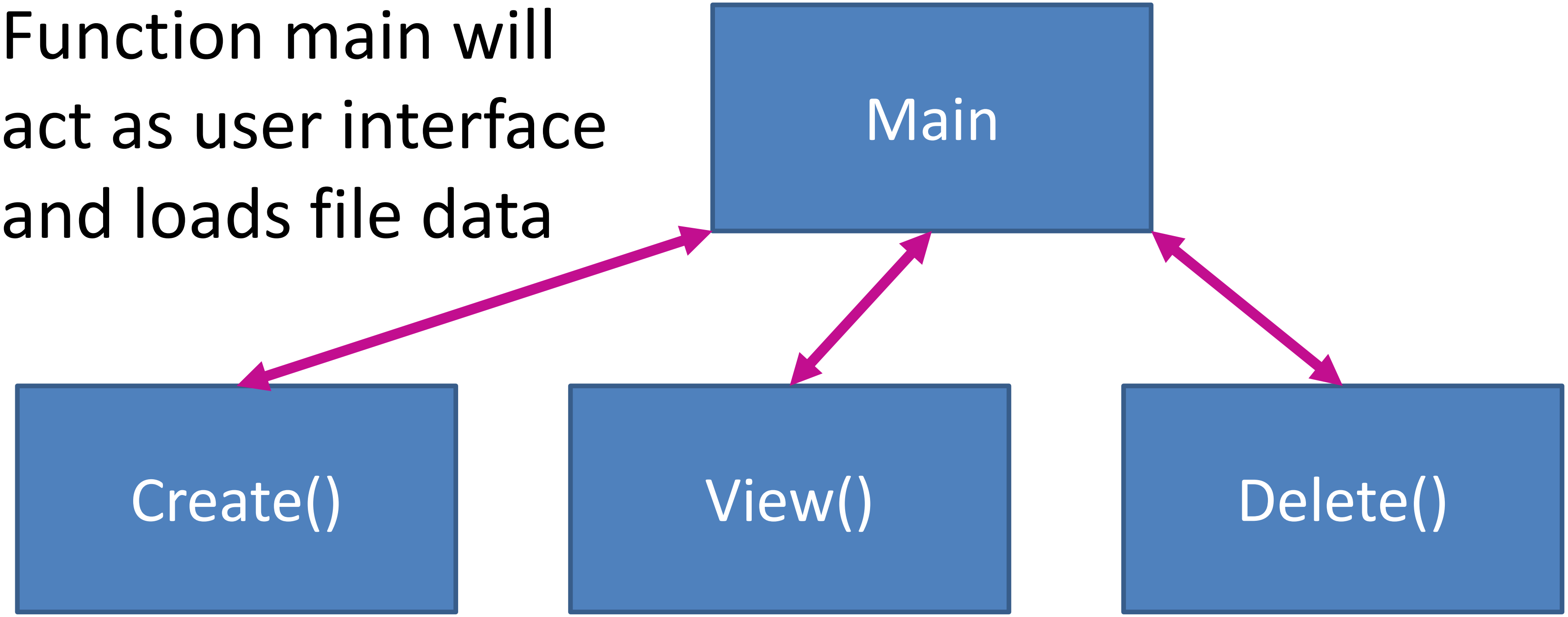
Phonebook - Array

Create data element

```
struct contact{
        char name[20];
        long long num;
        };
```

Phonebook - Array

Function main will act as user interface and loads file data



Need to determine how data flow will be managed between functions and within program.

Ideas?

Phonebook - Array

Filename to be stored in a variable

Global or local? Local

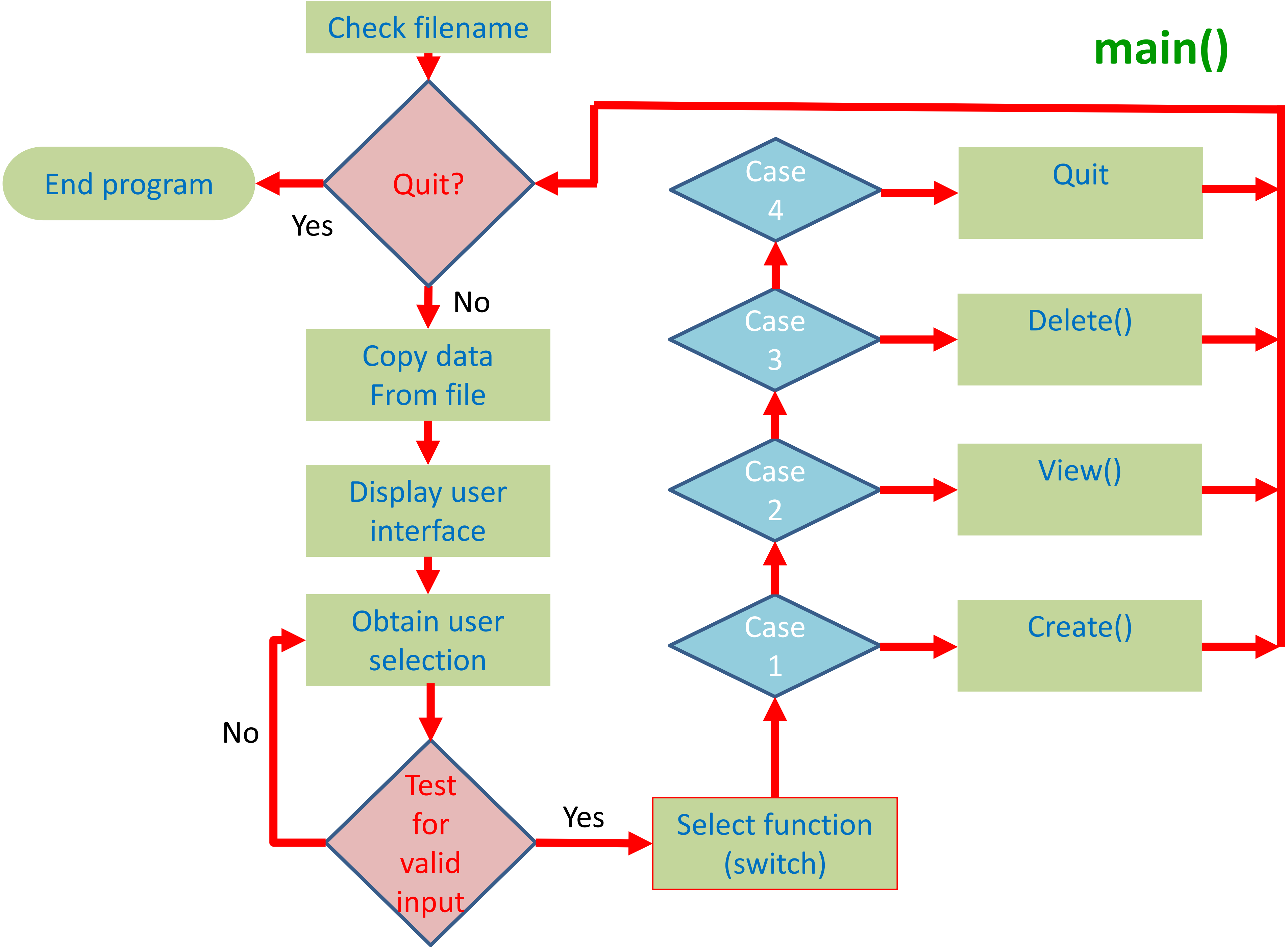
Whole file content to be read into array.

Global or local? Local

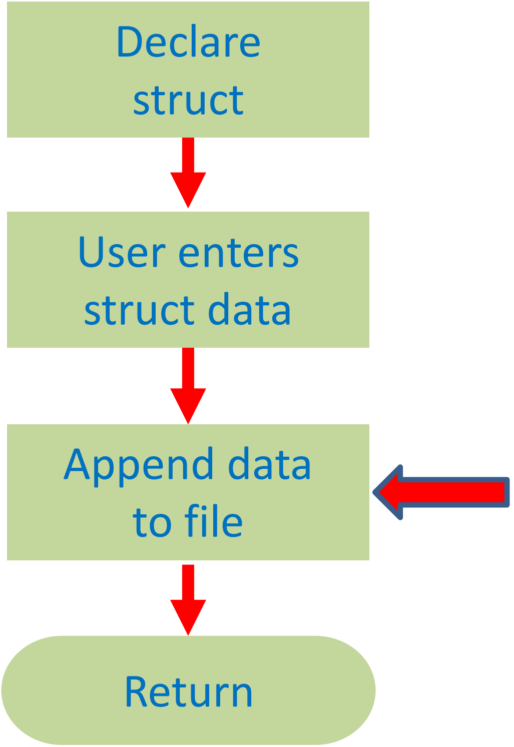
What information to be passed between functions?

- Number of elements in the array
- The array address
- The filename (depending on how functions work)

main()

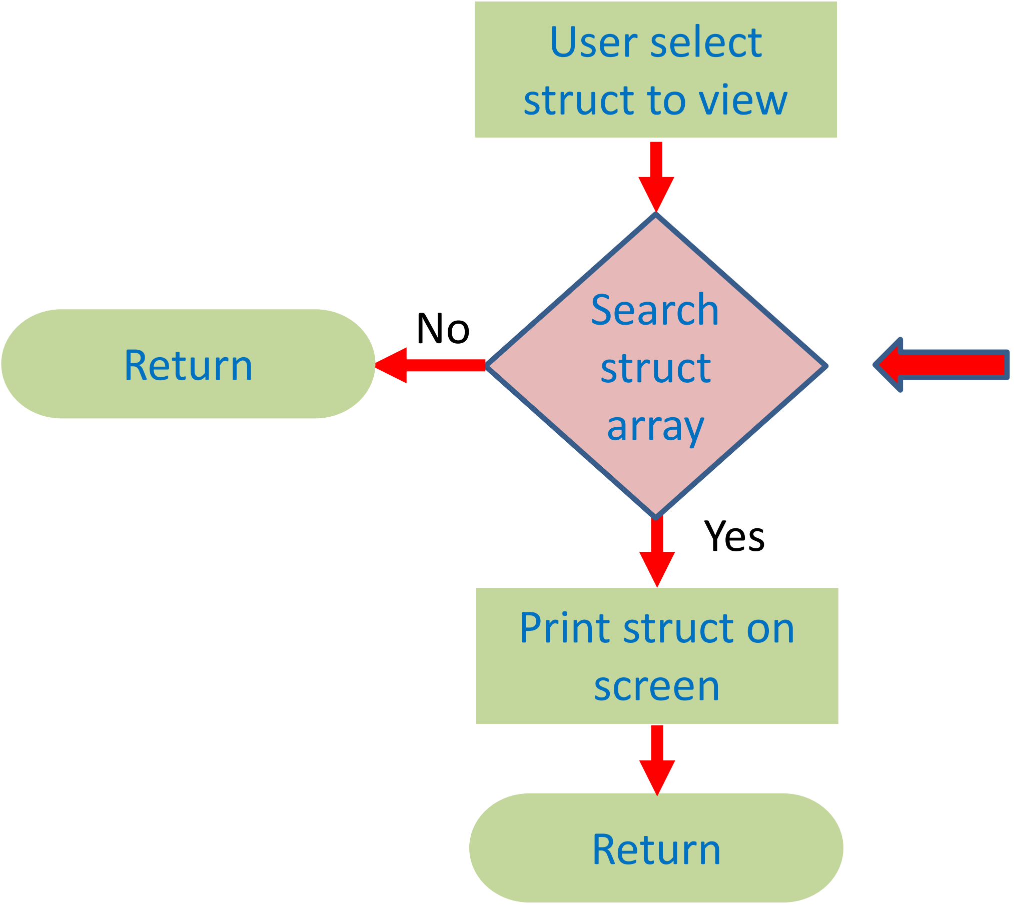

Check filename
Quit?
End program
Yes
No
Copy data From file
Display user interface
Obtain user selection
No
Test for valid input
Yes
Select function (switch)
Case 1
Case 2
Case 3
Case 4
Create()
View()
Delete()
Quit

Create()



Requires the filename for this

View()

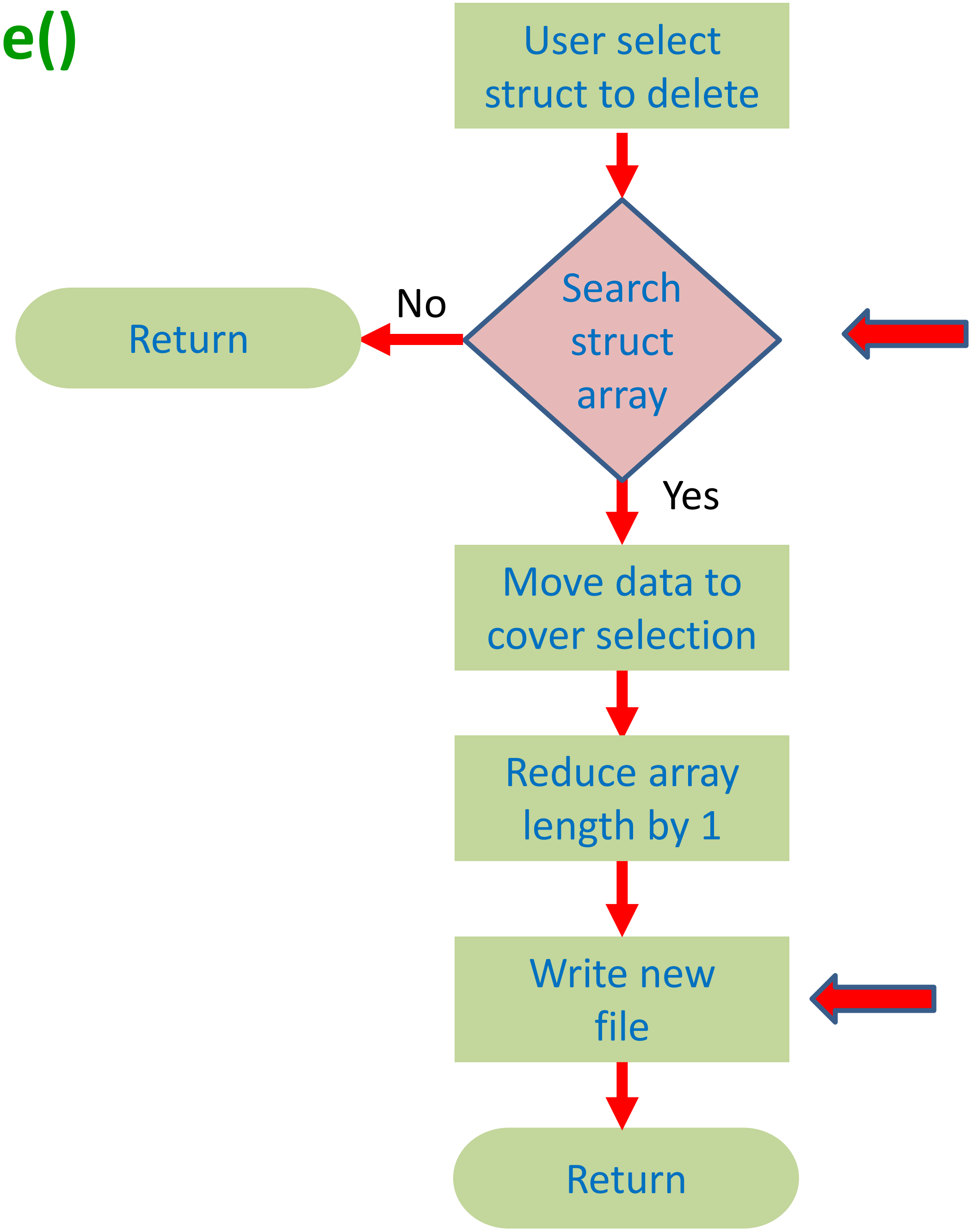


Requires the array of structures for this

Anything else?

Number of structures might be helpful

Delete()



Requires the array of structures for this

Also requires the number of structures

Requires the filename for this

Using one struct element

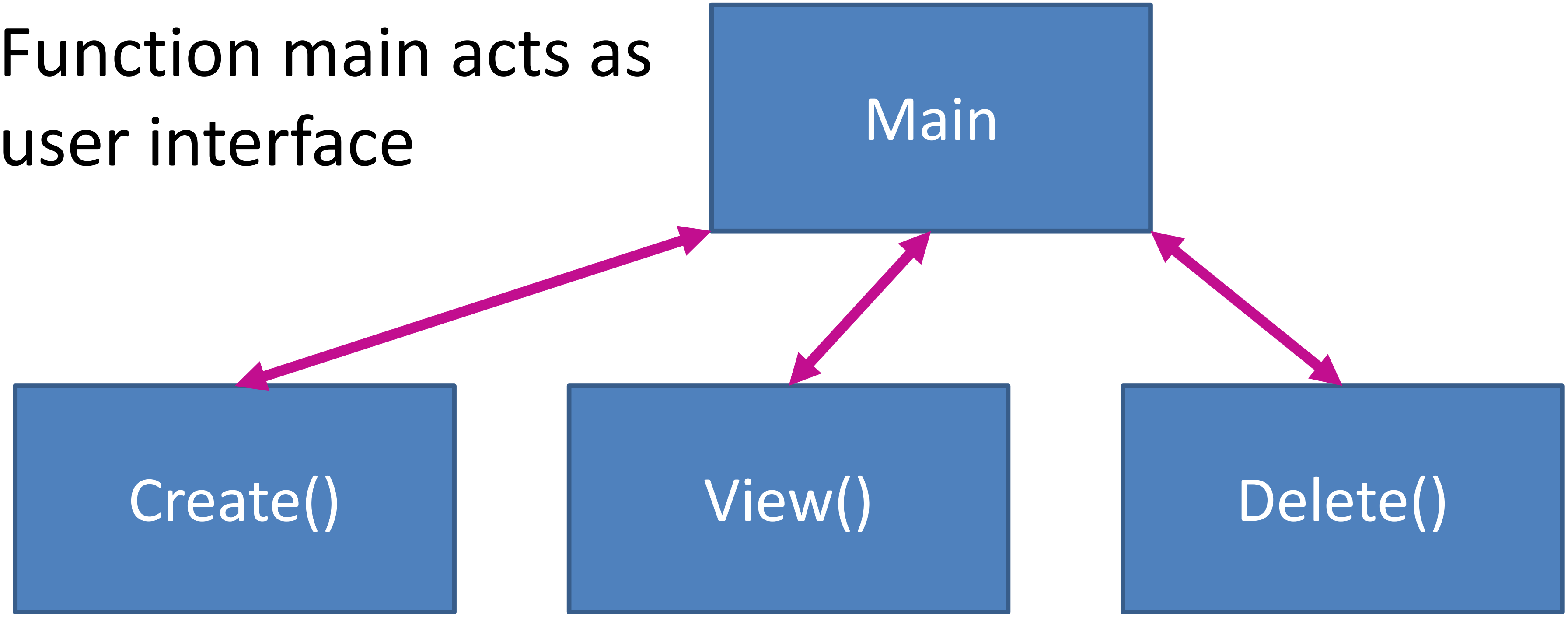
Phonebook - Element

Create data element

```
struct contact{
        char name[20];
        long long num;
        };
```

Phonebook - Array

Function main acts as user interface



Need to determine how data flow will be managed between functions and within program.

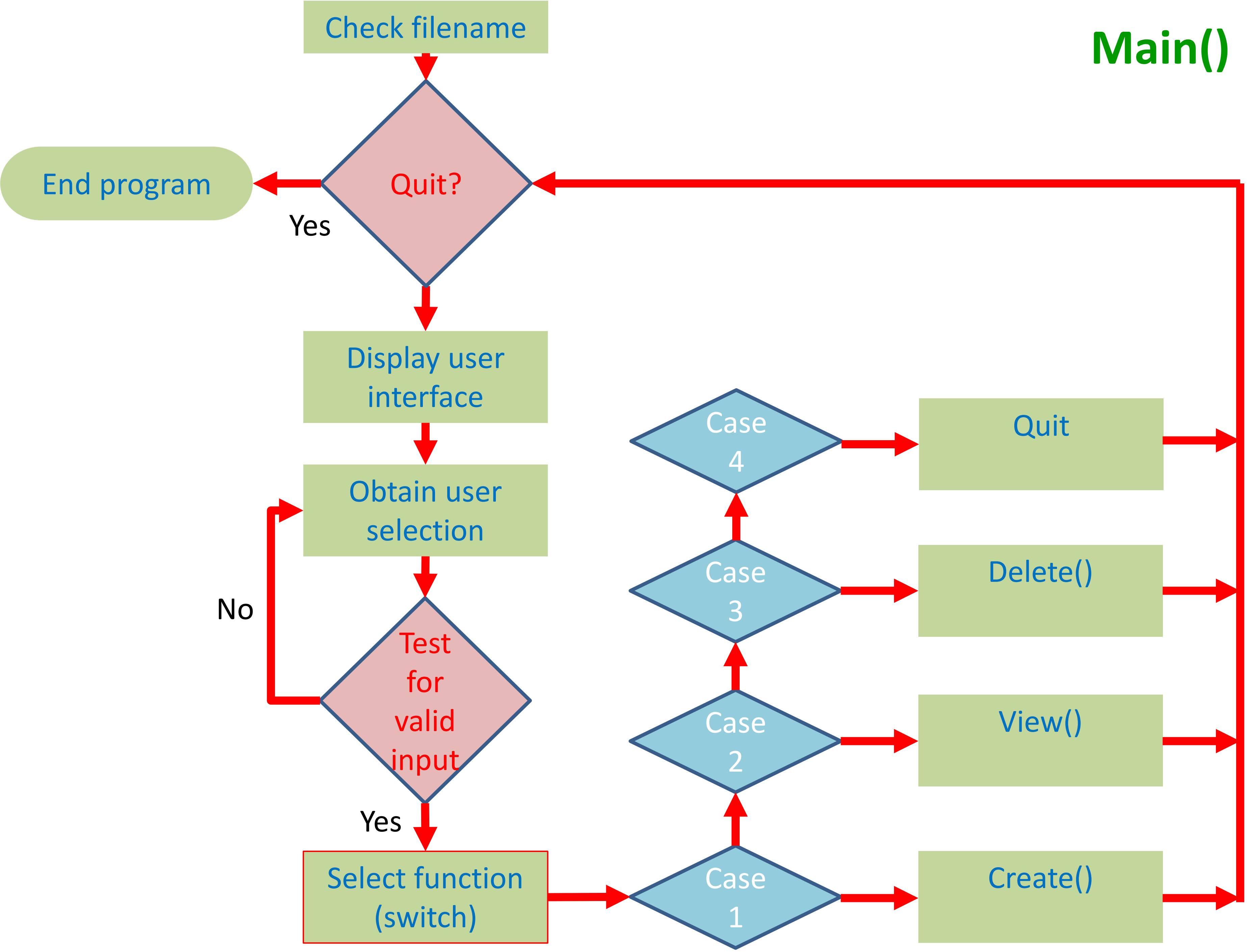
Phonebook – Single Element

Use all local variables

What information to be passed between functions?

- The filename

Main()



Check filename

Quit?

Yes

End program

Display user interface

Obtain user selection

Test for valid input

No

Yes

Select function (switch)

Case 1

Case 2

Case 3

Case 4

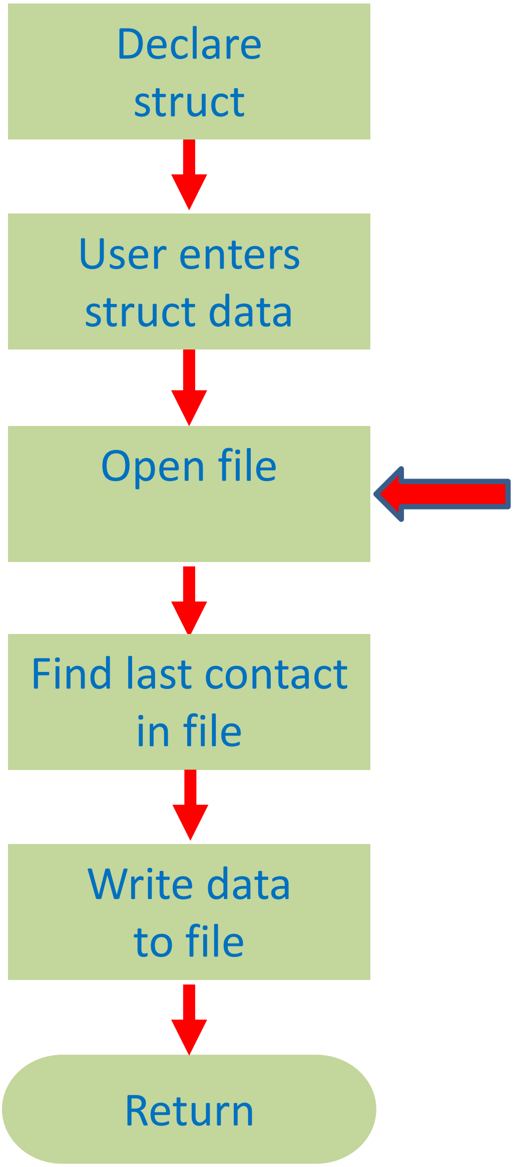
Create()

View()

Delete()

Quit

Create()



Declare struct

User enters struct data

Open file

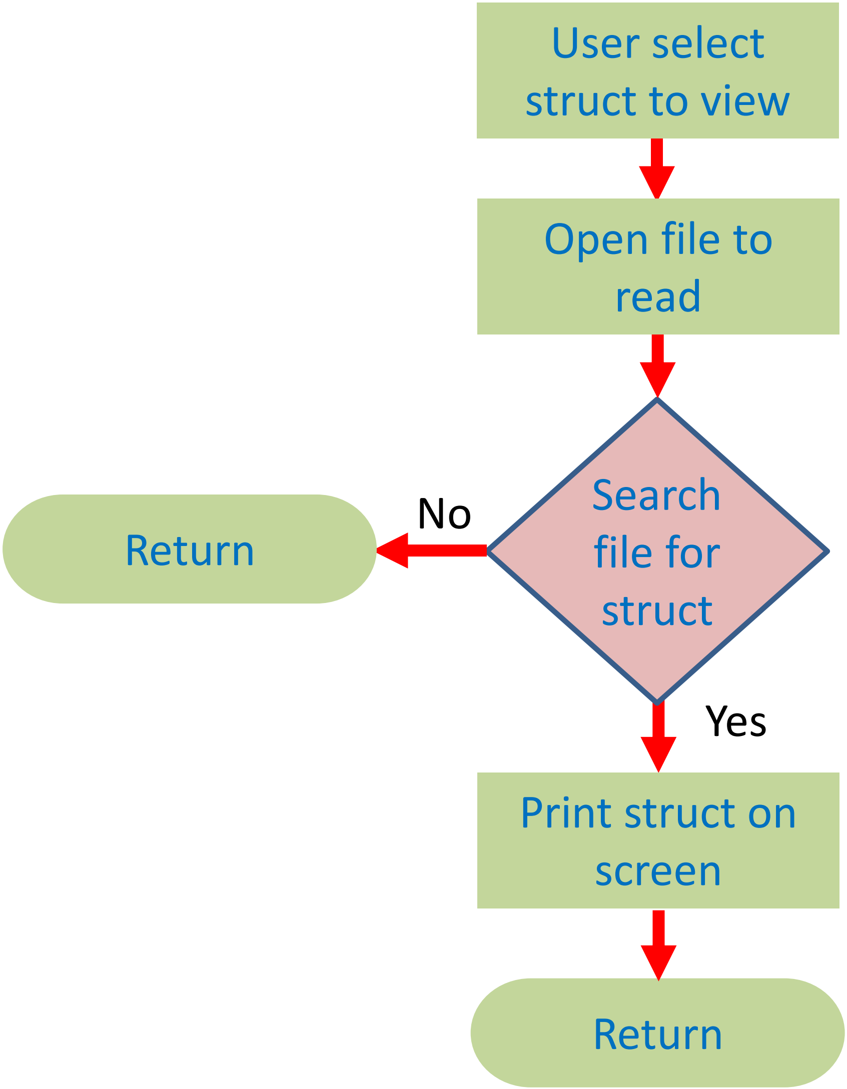
Find last contact in file

Write data to file

Return

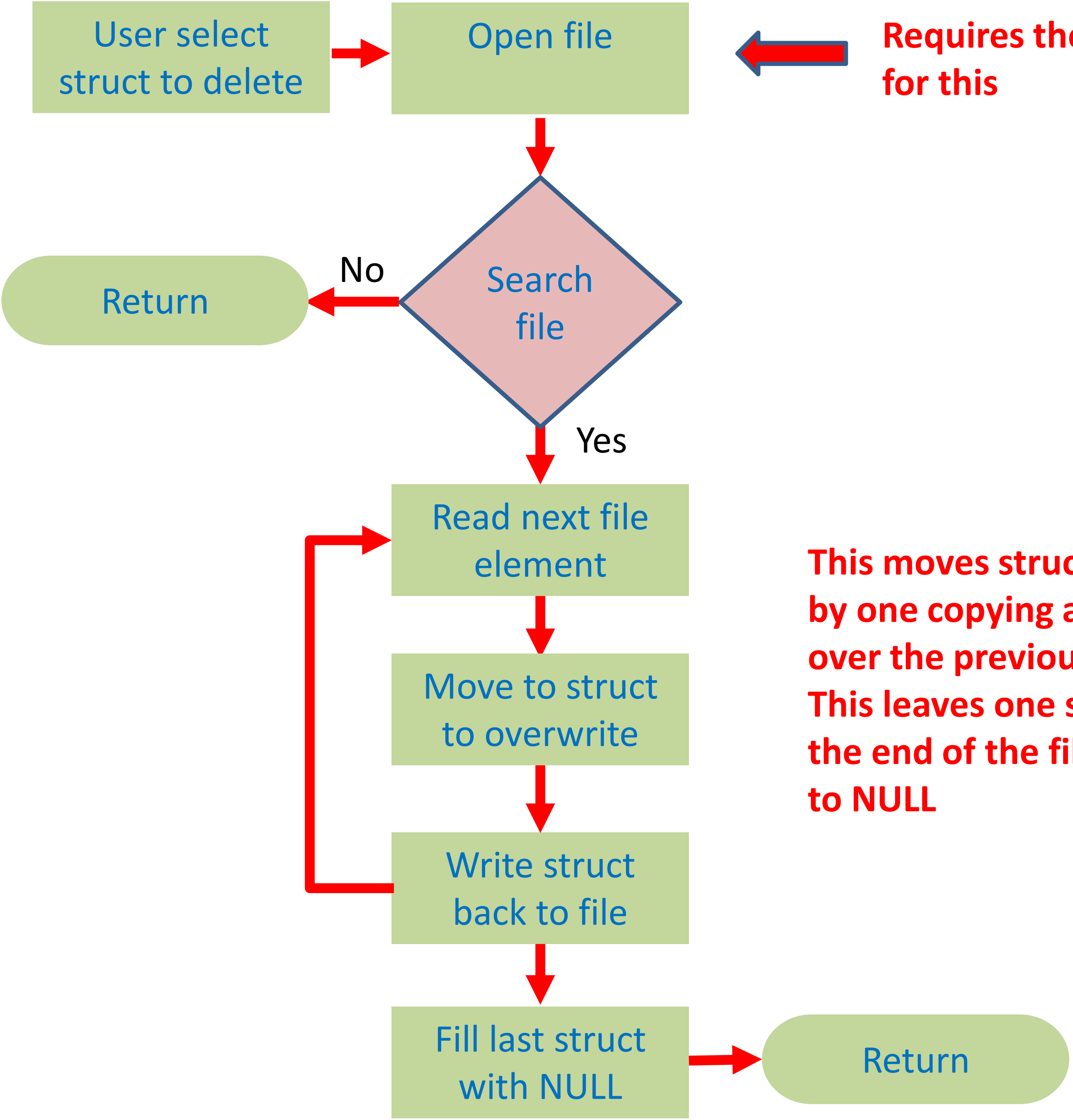
Requires the filename for this

View()



User select struct to view
Open file to read
Search file for struct
No
Return
Yes
Print struct on screen
Return

Delete()


User select struct to delete
Open file
Search file
No
Return
Yes
Read next file element
Move to struct to overwrite
Write struct back to file
Fill last struct with NULL
Return

Requires the filename for this

This moves structures one by one copying and writing over the previous structure. This leaves one structure at the end of the file to be set to NULL

Linked list

Phonebook – Linked List

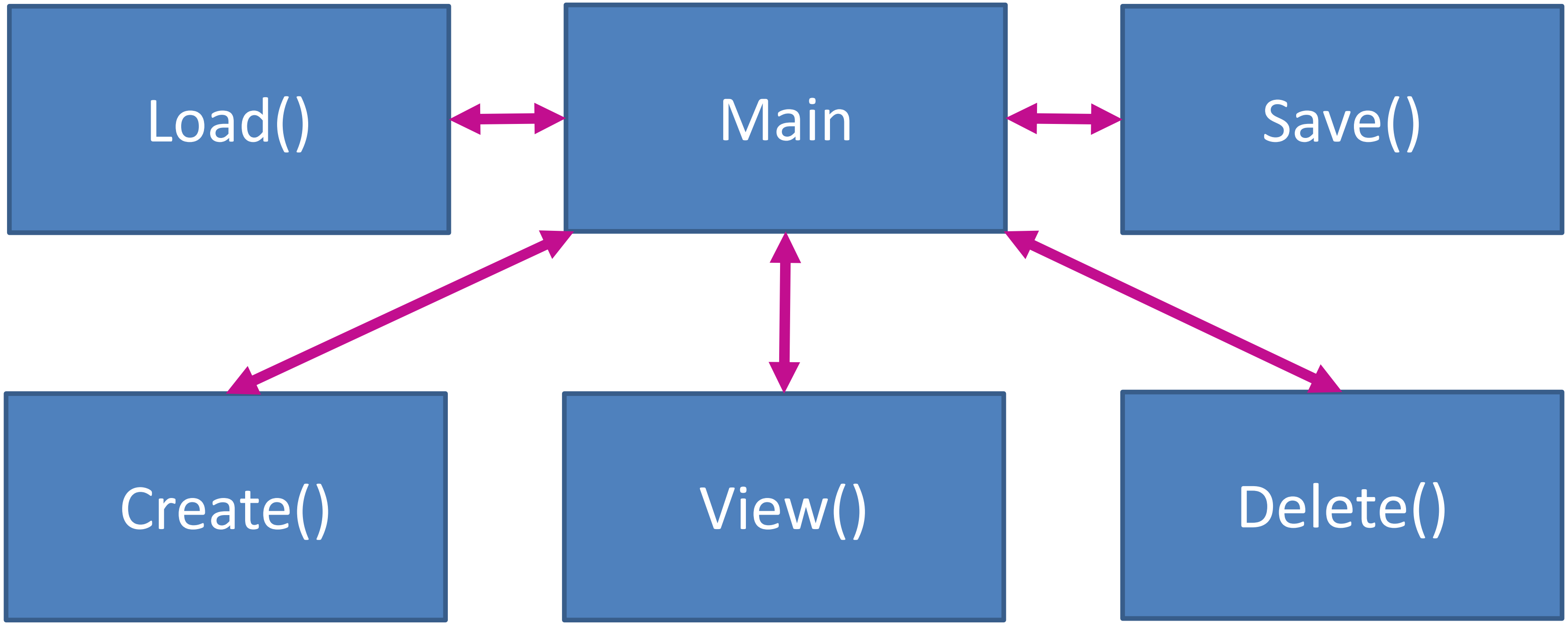
Create data element

```
struct contact{
        char name[20];
        long long num;
        struct contact *next;
        };
```

Additional pointer variable

Phonebook – Linked List

Function main
user interface



Need to determine how data flow will be managed between functions and within program.

Phonebook – Linked List

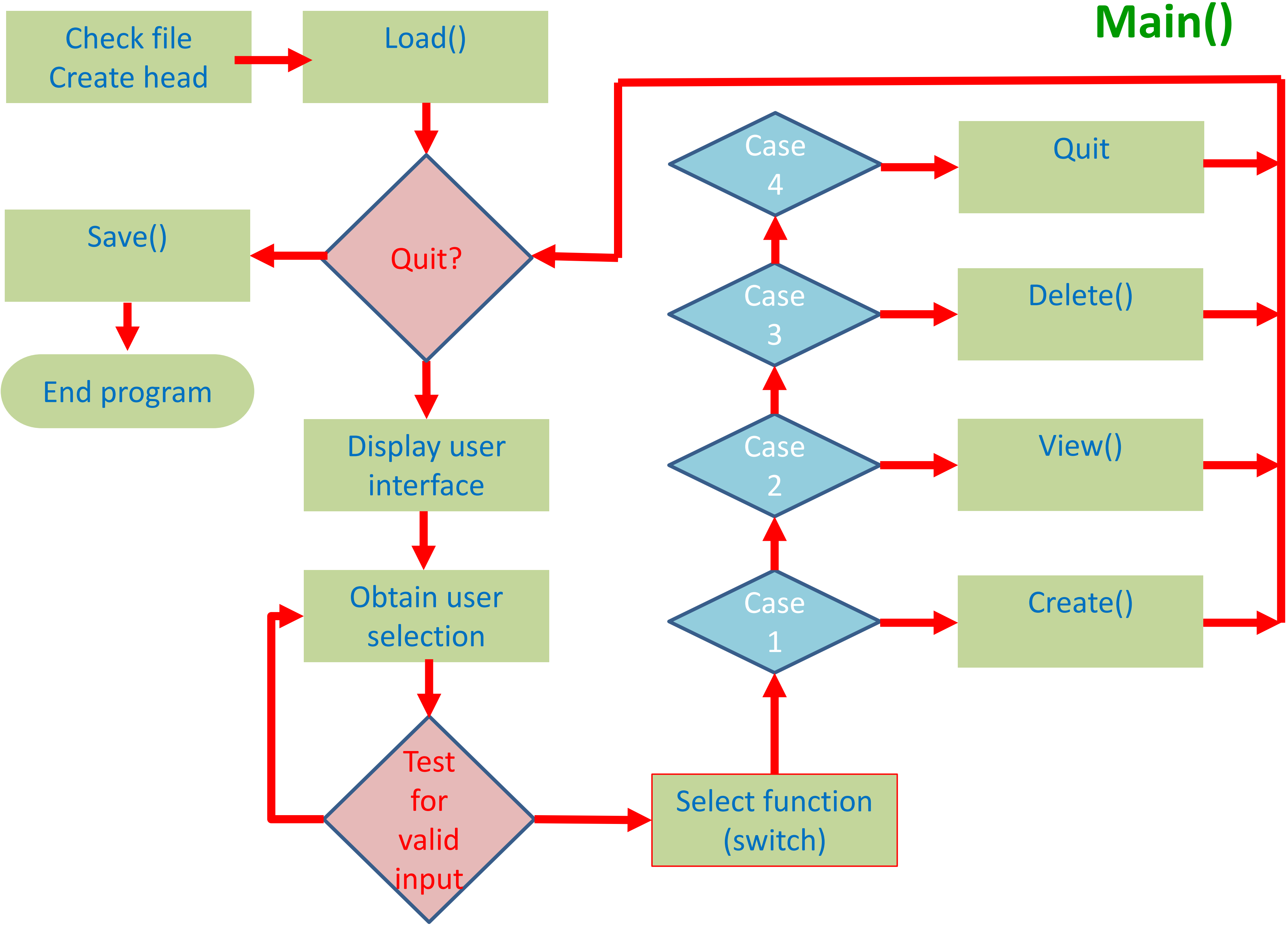
Use all local variables

Build linked list from file.

What information to be passed between functions?

- The head address of the list

Main()



Check file
Create head

Load()

Quit?

Save()

End program

Display user interface

Obtain user selection

Test for valid input

Select function (switch)

Case 1

Case 2

Case 3

Case 4

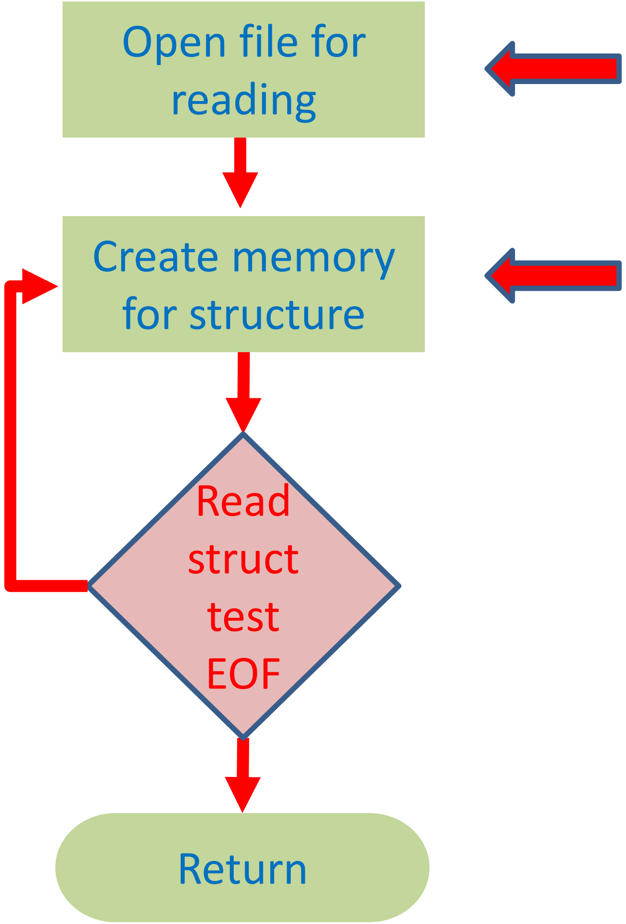
Create()

View()

Delete()

Quit

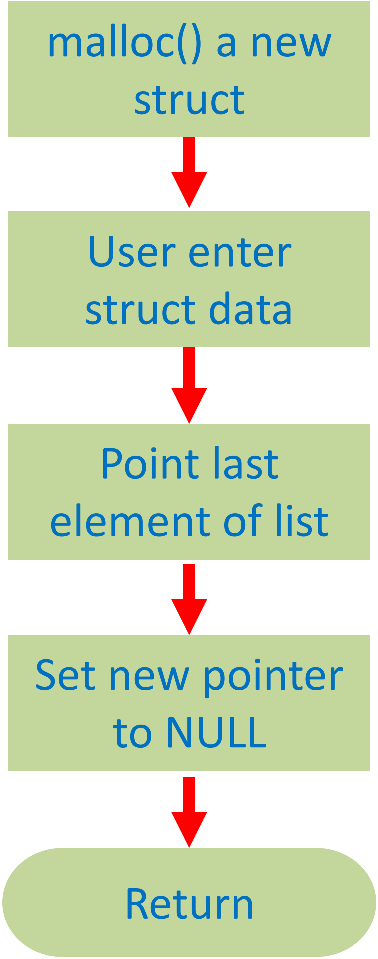
Load()



Requires the filename for this

Should start at the head… need to send that

Create()


malloc() a new struct

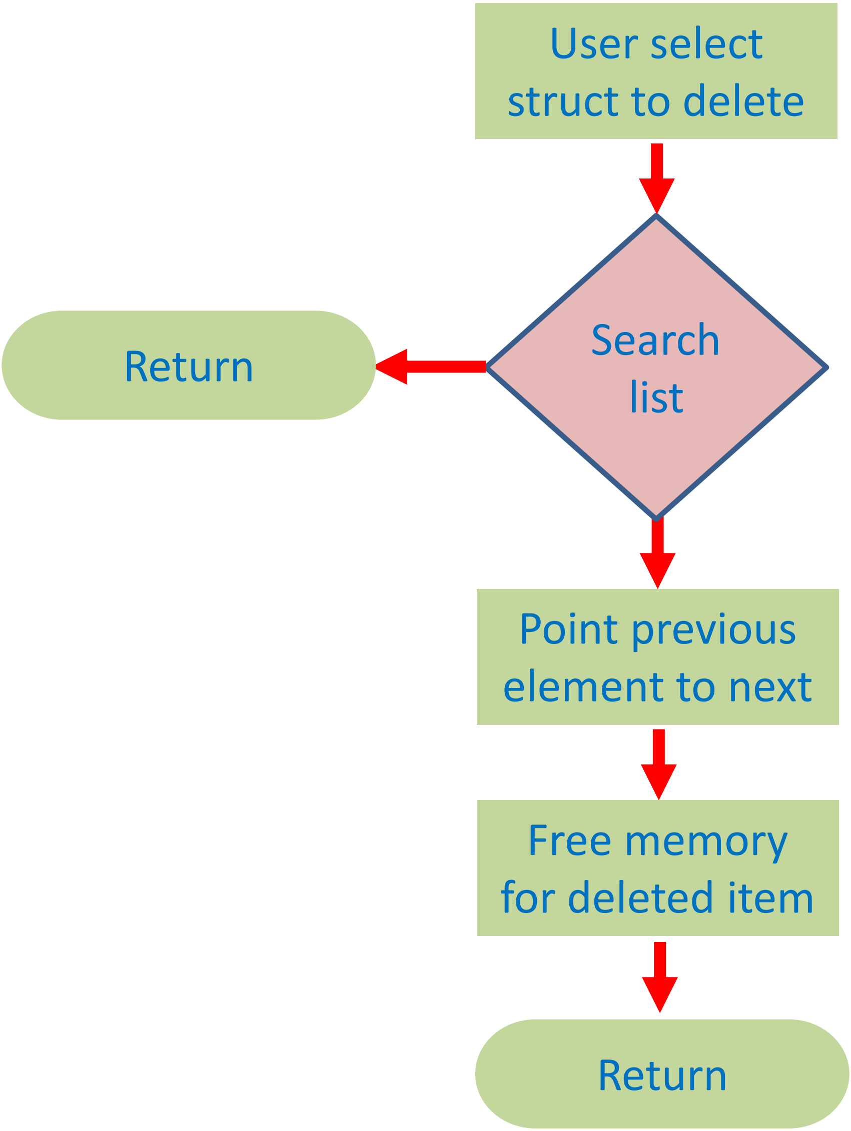
User enter struct data

Point last element of list

Set new pointer to NULL

Return

Delete()



User select struct to delete

Search list

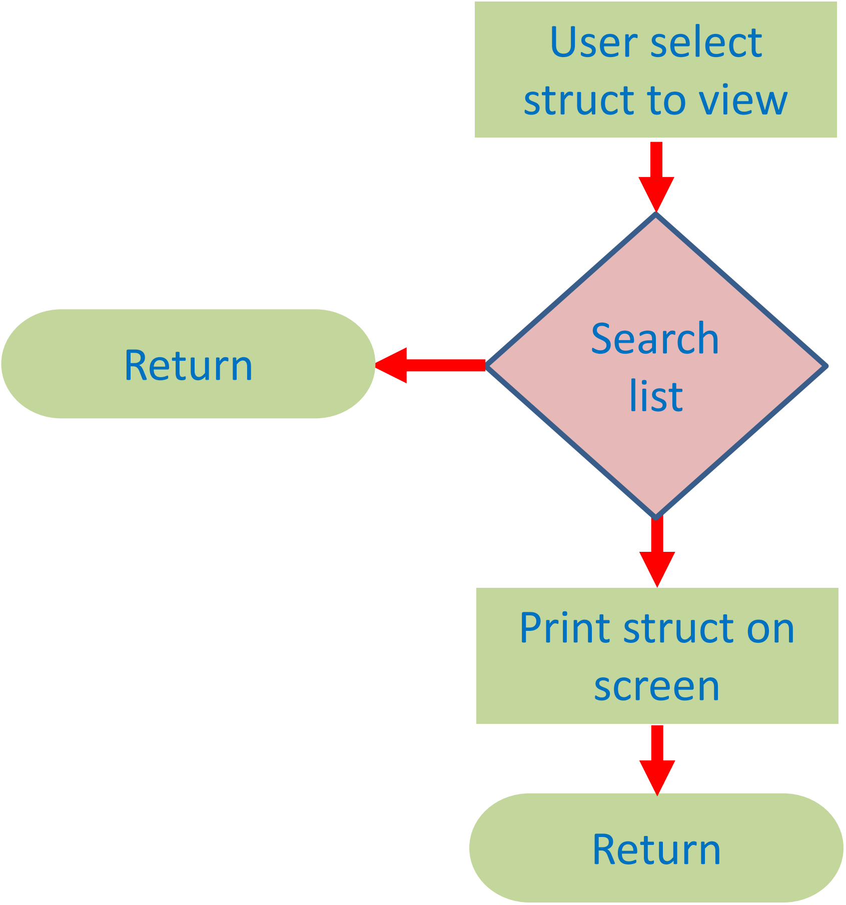
Return

Point previous element to next

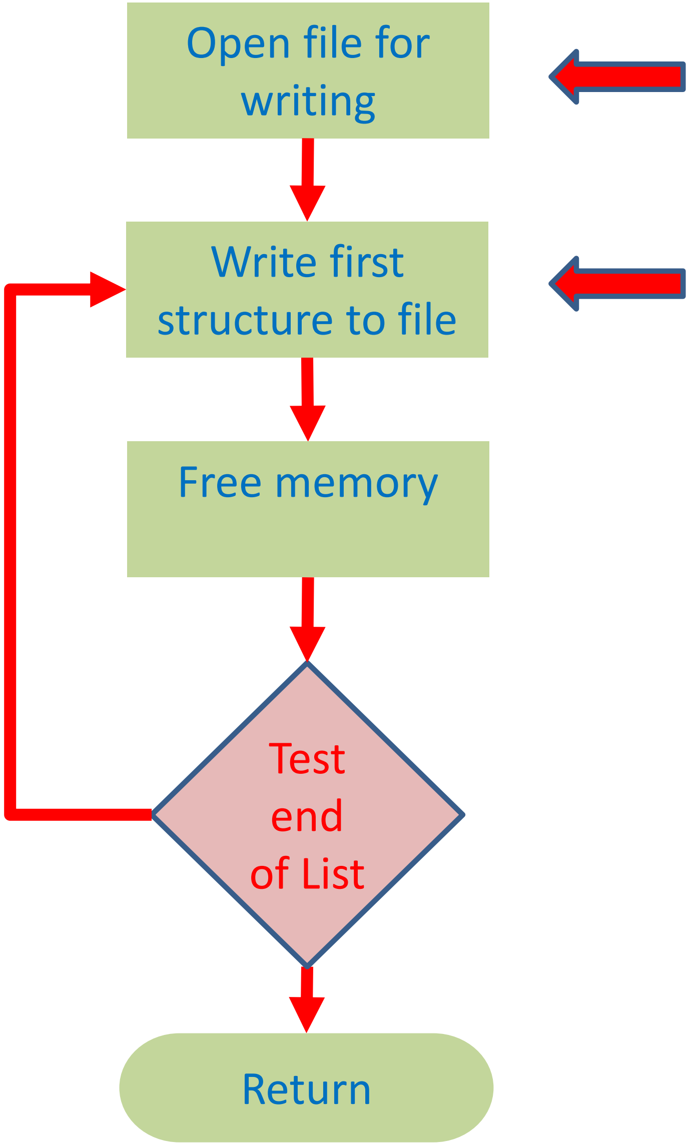
Free memory for deleted item

Return

View()


User select struct to view
Search list
Return
Print struct on screen
Return

Save()



Requires the filename for this

Should start at the head… need to send that

Questions?

The End… ish

but the labs are still going ☺