C Programming for Engineers

Data Structure

ICEN 360– Spring 2017

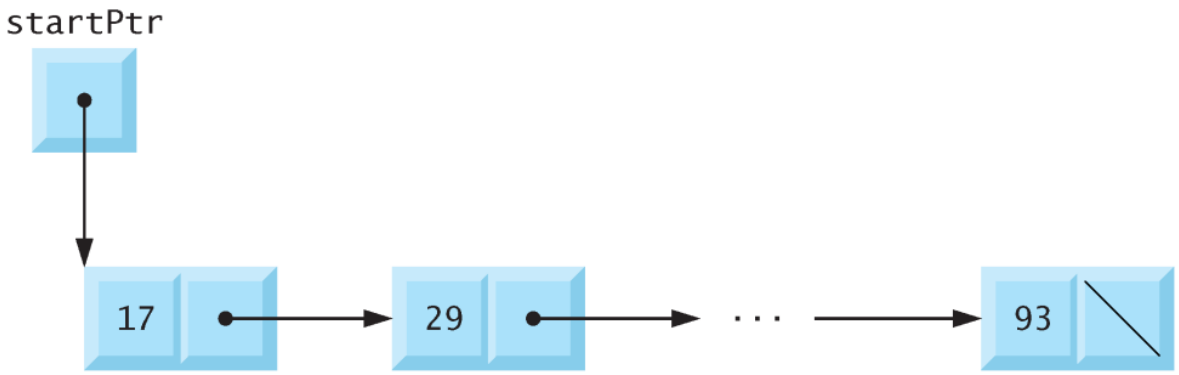
Prof. Dola Saha

Data Structures

- We've studied fixed-size data structures such as single-subscripted arrays, double-subscripted arrays and structs.
- This topic introduces dynamic data structures with sizes that grow and shrink at execution time.
  - Linked lists are collections of data items "lined up in a row"—insertions and deletions are made *anywhere* in a linked list.
  - Stacks are important in compilers and operating systems—insertions and deletions are made *only at one end* of a stack—its top.
  - Queues represent waiting lines; insertions are made *only at the back* (also referred to as the tail) of a queue and deletions are made *only from the front* (also referred to as the head) of a queue.
  - Binary trees facilitate high-speed searching and sorting of data, efficient elimination of duplicate data items, representing file system directories and compiling expressions into machine language.

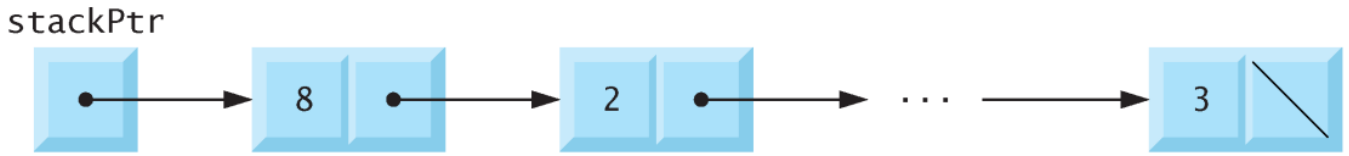
Linked List

- Linked lists are collections of data items "lined up in a row"—insertions and deletions are made *anywhere* in a linked list.
  - Linear Linked List
  - Doubly linked list
  - Circular linked list



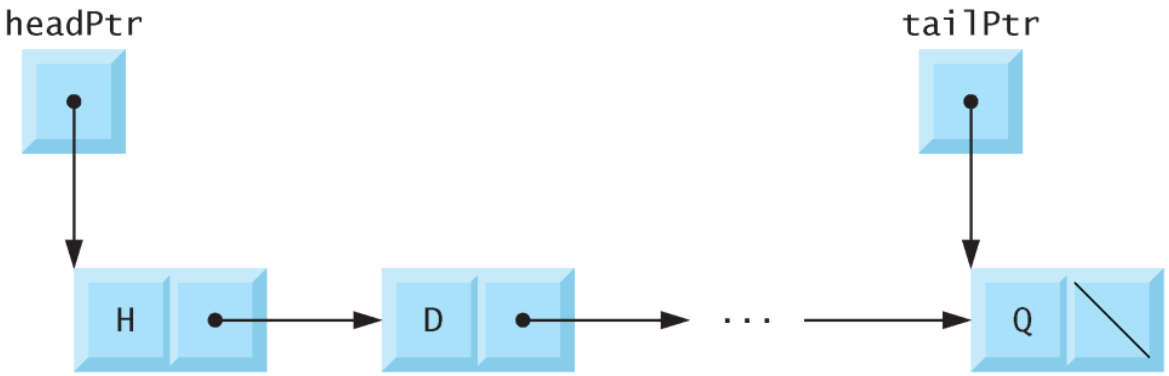
Stacks

- Stacks are important in compilers and operating systems—insertions and deletions are made *only at one end* of a stack—its top.
- Stack is referred to as LIFO (last-in-first-out).
- PUSH
- POP



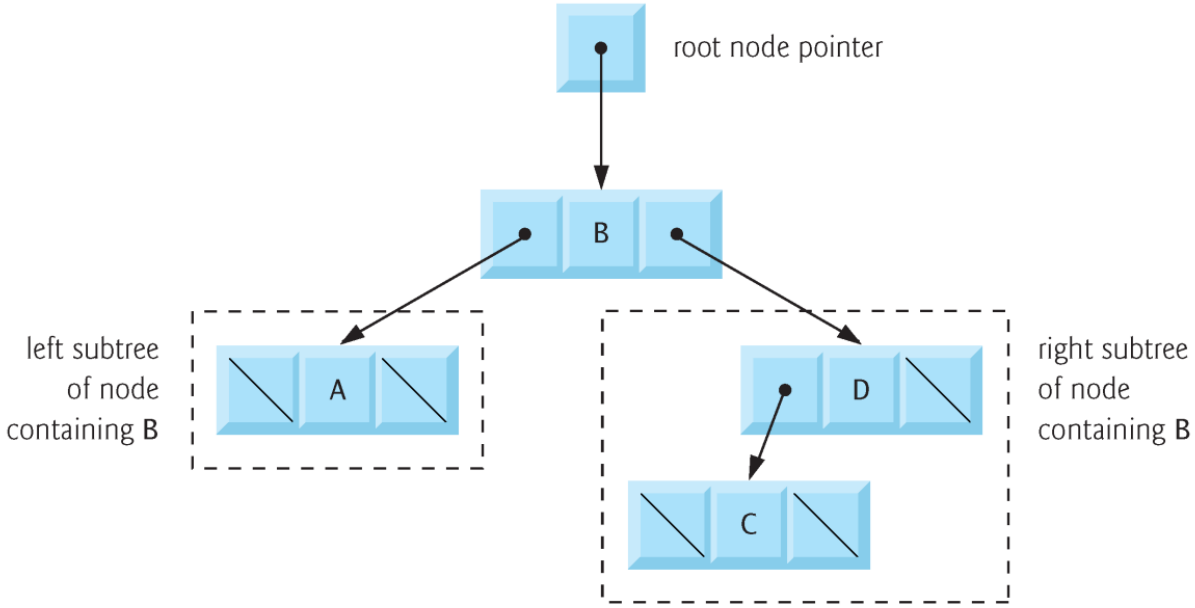
Queues

- Queues represent waiting lines; insertions are made *only at the back* (also referred to as the tail) of a queue and deletions are made *only from the front* (also referred to as the head) of a queue.
- Used in networking when packets are queued to move from one layer to another.
- Enqueue
- Dequeue



Trees

- A tree is a *nonlinear, two-dimensional data structure* with special properties.
- Tree nodes contain *two or more* links.
- Binary trees facilitate high-speed searching and sorting of data, efficient elimination of duplicate data items, representing file system directories and compiling expressions into machine language.



Dynamic Memory Allocation

- Creating and maintaining dynamic data structures requires dynamic memory allocation—the ability for a program to *obtain more memory space at execution time* to hold new nodes, and to *release space no longer needed*.
- Functions malloc and free, and operator sizeof, are essential to dynamic memory allocation.

malloc()

- *void* * **malloc** *(size_t size)*
- Input: number of bytes to be allocated
- Output: a pointer of type void * (pointer to void) to the allocated memory.
- A void * pointer may be assigned to a variable of *any* pointer type.
- Example:

```
newPtr = malloc(sizeof(int));
```

- The allocated memory is *not* initialized.
- If no memory is available, malloc returns NULL.

free()

- Function free *deallocates* memory—i.e., the memory is *returned* to the system so that it can be reallocated in the future.
- To *free* memory dynamically allocated by the preceding malloc call, use the statement
  - `free(newPtr);`
- C also provides functions calloc and realloc for creating and modifying *dynamic arrays*.

Dynamic Memory Allocation

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
   char *str;

   /* Dynamic Memory allocation */
   str = (char *) malloc(50);
   if (str == NULL)
   {
       printf("Error in memory allocation.");
       return(1);
   }

   strcpy(str, "Programming is fun!");
   printf("String = %s\n", str);

   // Free the memory allocated
   free(str);

   return(0);
}
```

Classroom Assignment

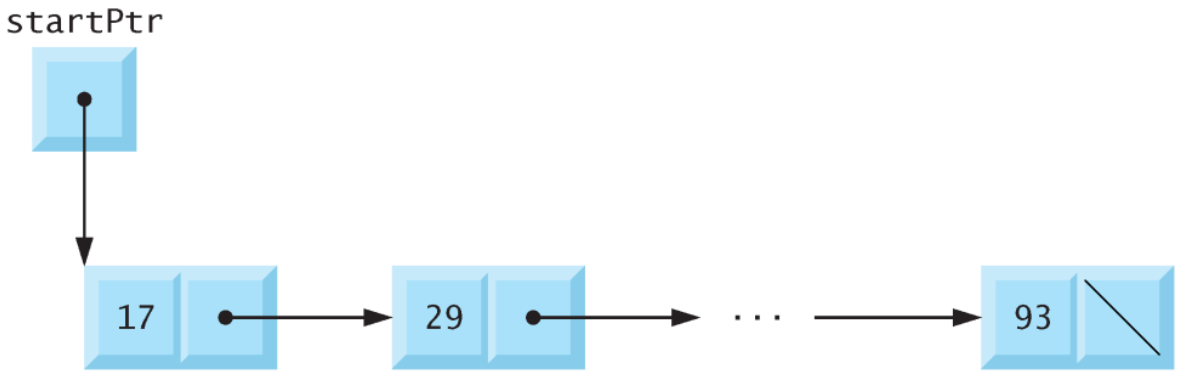
- Create a dynamic array of N elements, where the user may choose the elements to be either integer, float or double. Once the array is created, fill the array with random numbers. Finally, print out the values.

Self Referencing Structures

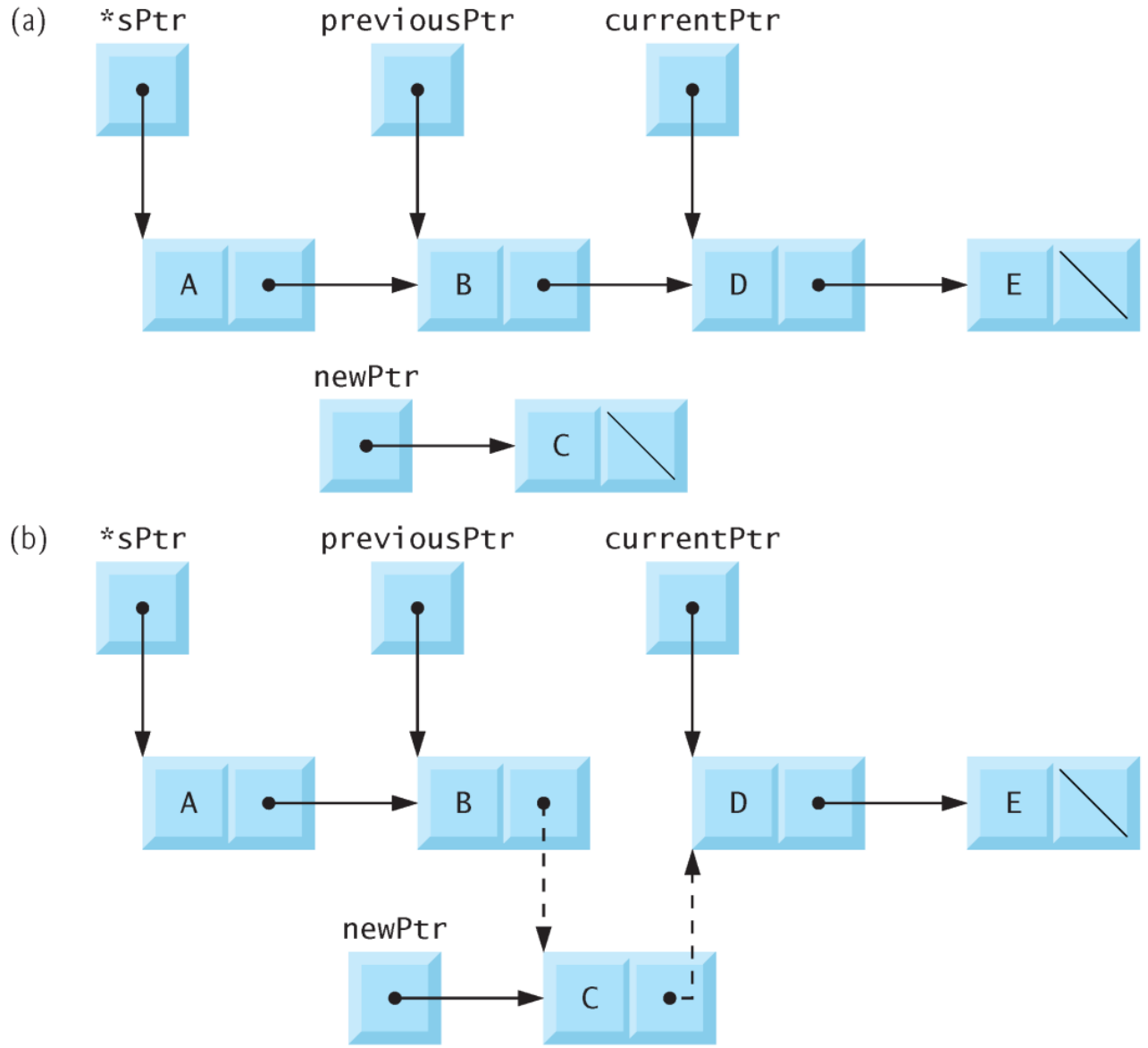
- A *self-referential structure* contains a pointer member that points to a structure of the *same* structure type.
- Example:
  - ```
    struct node {
        int data;
        struct node *nextPtr;
    };
    ```

  defines a type, `struct node`.
- A structure of type `struct node` has two members – integer member `data` and pointer member `nextPtr`.
    ```

Linked List graphical representation



Insert a node in order in a list



Insert a node – C code

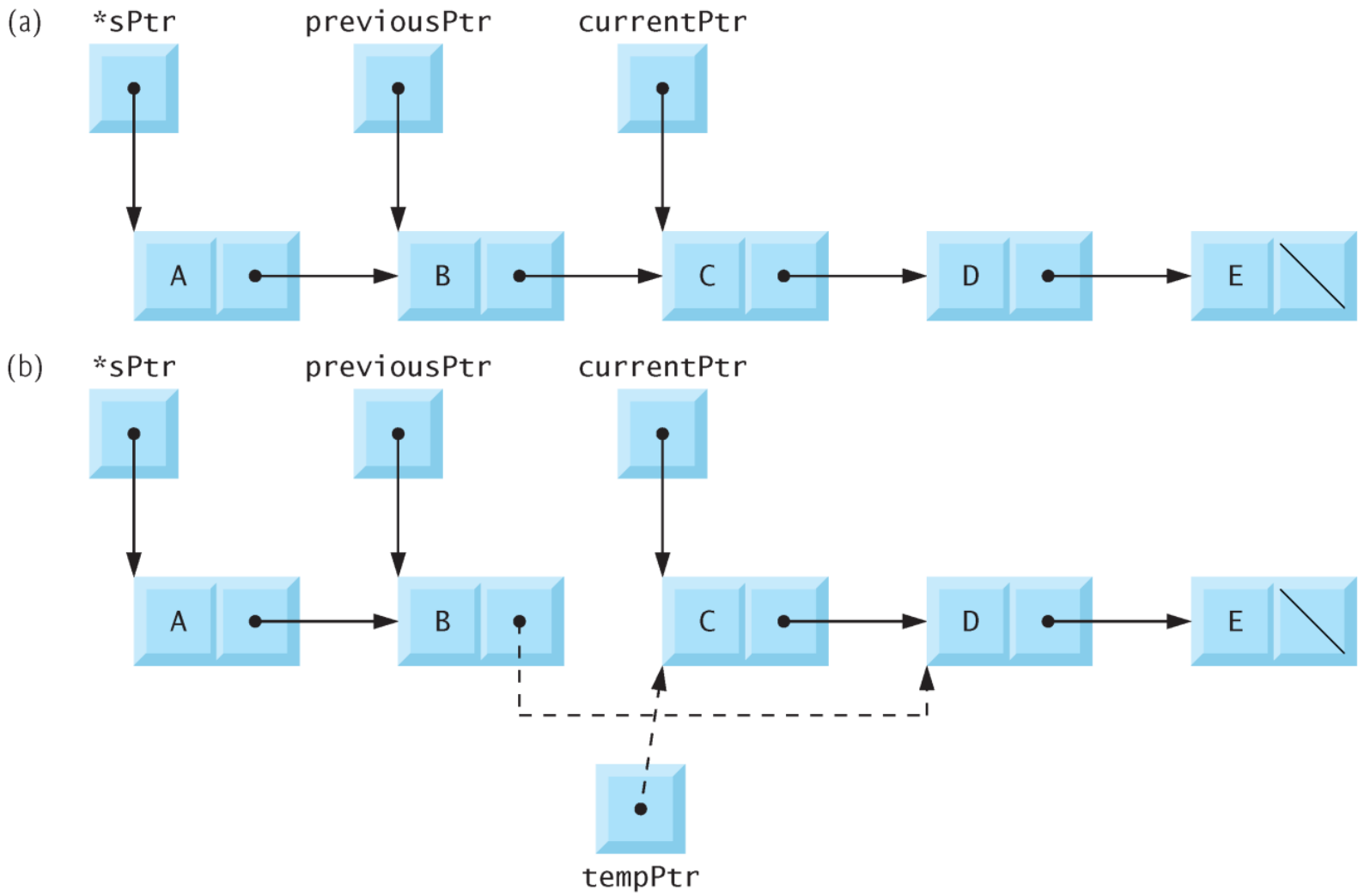
```
void insert(ListNodePtr *sPtr, char value)
{
   ListNodePtr newPtr = malloc(sizeof(ListNode)); // create node

   if (newPtr != NULL) { // is space available?
      newPtr->data = value; // place value in node
      newPtr->nextPtr = NULL; // node does not link to another node

      ListNodePtr previousPtr = NULL;
      ListNodePtr currentPtr = *sPtr;
      // loop to find the correct location in the list
      while (currentPtr != NULL && value > currentPtr->data) {
         previousPtr = currentPtr; // walk to ...
         currentPtr = currentPtr->nextPtr; // ... next node
      }

      // insert new node at beginning of list
      if (previousPtr == NULL) {
         newPtr->nextPtr = *sPtr;
         *sPtr = newPtr;
      }
      else { // insert new node between previousPtr and currentPtr
         previousPtr->nextPtr = newPtr;
         newPtr->nextPtr = currentPtr;
      }
   }
   else {
      printf("%c not inserted. No memory available.\n", value);
   }
}
```

Delete a node from list



Delete a node – C code

```
// delete a list element
char delete(ListNodePtr *sPtr, char value)
{
   // delete first node if a match is found
   if (value == (*sPtr)->data) {
      ListNodePtr tempPtr = *sPtr; // hold onto node being removed
      *sPtr = (*sPtr)->nextPtr; // de-thread the node
      free(tempPtr); // free the de-threaded node
      return value;
   }
   else {
      ListNodePtr previousPtr = *sPtr;
      ListNodePtr currentPtr = (*sPtr)->nextPtr;

      // loop to find the correct location in the list
      while (currentPtr != NULL && currentPtr->data != value) {
         previousPtr = currentPtr; // walk to ...
         currentPtr = currentPtr->nextPtr; // ... next node
      }

      // delete node at currentPtr
      if (currentPtr != NULL) {
         ListNodePtr tempPtr = currentPtr;
         previousPtr->nextPtr = currentPtr->nextPtr;
         free(tempPtr);
         return value;
      }
   }

   return '\0';
}
```

Classwork Assignment

- Write a function to print the elements of a list.