CSE 1320

Week of 04/29/2019

Instructor : Donna French

```
/* QueueLib For Coding Assignment 7 */

#ifndef _QUEUE_LIB_H
#define _QUEUE_LIB_H

typedef struct QNODE
{
    char Name[30];
    struct QNODE *next_ptr;
} QNODE;

// First parameter - Customer's name
// Second parameter - Address of QueueHead
// Third parameter - Address of QueueTail
void enQueue(char [], QNODE **, QNODE **);

// First parameter - Address of QueueHead
void deQueue(QNODE **);

// First parameter - QueueHead
void DisplayQueue(QNODE *);

#endif
```

```
typedef struct BNODE
{
    char MovieTheaterName[40];
    char ZipCode[6];
    char FileName[100];
    char Dimensions[6];
    struct BNODE *left;
    struct BNODE *right;
} BNODE;

// First parameter - Address of BSTnode
// Second parameter - MovieTheaterName
// Third parameter - ZipCode
// Fourth parameter - FileName containing seat map (XXXOOOO)
// Fifth parameter - Dimensions of theater
void AddBSTNode(BNODE **, char [], char [], char [], char []);

// First parameter - BST node
void InOrder(BNODE *);

// First parameter - BST node
// Second parameter - zipcode of movie theater being searched for
BNODE *SearchForBNODE(BNODE *, char []);
```

```
/* ListLib For Coding Assignment 7 */

#ifndef _LIST_LIB_H
#define _LIST_LIB_H

typedef struct LNODE
{
    char Ticket[3];
    struct LNODE *next_ptr;
} LNODE;

// First parameter - Address of linked list head
// Second parameter - Ticket being added to linked list
void InsertNode(LNODE **, char[]);

// First parameter - Address of linked list head
// Second parameter - One ticket that has been taken off the linked list (node was
freed)
void ReturnAndFreeLinkedListNode(LNODE **, char []);

#endif
```

```
#include "ListLib.h"

#ifndef _STACK_LIB_H
#define _STACK_LIB_H

typedef struct SNODE
{
    int ReceiptNumber;
    char MovieTheaterName[30];
    LNODE *TicketList;
    struct SNODE *next_ptr;
} SNODE;

// First parameter - Address of StackTop
// Second parameter - Head of ticket linked list stored in stack node
// Third parameter - Receipt number stored in stack node
// Fourth parameter - Movie theater name stored in stack node
void push(SNODE **, LNODE *, int, char[]);

// First parameter - Address of StackTop
void pop(SNODE **);

#endif
```

Write a recursive function that sums of all natural numbers from a value n to 1

So if n is 5, we would start with 5 and then add 4 and 3 and 2 and 1.

5 + 4 + 3+ 2 + 1 = 15

So if n is 5, then 4 is n-1.
When n is 4, then 3 is n-1.
When n is 3, then 2 is n-1.
When n is 2, then 1 is n-1.
When n is 1, then 0 is n-1.

We stop at 0/after we reach 1

The recursive call is n + (n – 1)

Write a recursive function that sums of all natural numbers from a value X to 1

```
int addNumbers(int n)
{
    if (n != 0)
    {
        return n + addNumbers(n-1);
    }
    else
    {
        return n;
    }
}
```

Given a binary tree that already exists and has a node definition of

```
typedef struct BNODE
{
   int MyValue;
   struct BNODE *left_ptr;
   struct BNODE *right_ptr;
} BNODE;
```

Write a recursive function to count the number of nodes in the tree.

Given a binary tree that already exists and has a node definition of

```
typedef struct BNODE
{
    int MyValue;
    struct BNODE *left_ptr;
    struct BNODE *right_ptr;
} BNODE;
```

Write a recursive function to sum `MyValue` from all nodes in the tree.

Given a linked list with head LLH1 and a linked list with head LLH2, write a function to concatenate LLH2 onto the end of LLH1.

Given a stack containing data, write the function to reverse the order of the nodes in the stack.

Given two queues, write a function to evenly merge them into a new queue.

Conditional Compilation

Conditional compilation allows a choice of lines to be compiled based on the definition of certain symbols at compile time.

The preprocessor handles processing these lines.

We are familiar with using `#define` to associate symbols with values

```
#define TRUE 1
#define FALSE 0
```

This sets the symbol `TRUE` to `1` and the symbol `FALSE` to `0`.

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

#define TRUE 1
#define FALSE 0

int main(void)
{
     if (TRUE)
          printf("HI");
     else if (FALSE)
          printf("BYE");

     return 0;
}
```

```
gcc concom2Demo.c -E

int main(void)
{
 if (1)
  printf("HI");
 else if (0)
  printf("BYE");

 return 0;
}
```

Compiler option to just run the preprocessor

concom2Demo.c

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

#define TRUE FROG
#define FALSE TOAD

int main(void)
{
      if (TRUE)
            printf("HI");
      else if (FALSE)
            printf("BYE");

      return 0;
}
```

```
gcc concom2Demo.c -E
```

```
int main(void)
{
 if (FROG)
  printf("HI");
 else if (TOAD)
  printf("BYE");

 return 0;
}
```

```
concom2Demo.c: In function ‘main’:
concom2Demo.c:10: error: ‘FROG’ undeclared (first use in this
function)
concom2Demo.c:10: error: (Each undeclared identifier is reported
only once
concom2Demo.c:10: error: for each function it appears in.)
concom2Demo.c:12: error: ‘TOAD’ undeclared (first use in this
function)
```

Conditional Compilation

`#define`	used to define symbols to the compiler
`#ifdef`	tests whether a symbol is defined to the compiler
`#else`	provides an alternative to `#ifdef`
`#elif`	used to build compound conditional directives
`#endif`	signals the end of the body of an if directive
`#ifndef`	tests whether a symbol is not defined
`#if`	tests values of constant expressions

Conditional Compilation

`#if 0` and `#endif` can be used to comment out large chunks of code.

```
#if 0
printf(“Hello World”);
I need to add a line here to set x = 3
#endif
```

This a good way to comment out large sections of code while you are still working on your program and have parts that are unfinished. Allows you to compile what you have working.

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

#define TRUE 1
#define FALSE 0

int main(void)
{
#if 0
     if (TRUE)
          printf("HI");
     else if (FALSE)
          printf("BYE");
#endif
     return 0;
}
```

```
int main(void)
{








 return 0;
}
```

concom3Demo.c

Conditional Compile

Without `include` guard

```
HELLO
HELLO

int main(void)
{
    return 0;
}
```

`gcc concom4Demo.c -E`

```
HELLO

HELLO

int main(void)
{
 return 0;
}
```

Conditional Compile

With `include` guard

```
#ifndef INCLUDE_GUARD
#define INCLUDE_GUARD
HELLO
#endif

#ifndef INCLUDE_GUARD
#define INCLUDE_GUARD
HELLO
#endif


int main(void)
{
   return 0;
}
```

`gcc concom4Demo.c –E`

```
HELLO


int main(void)
{
   return 0;
}
```

Conditional Compile

```
#define DEBUGS

#ifdef DEBUGS
    printf("We are debugging");
#endif
```

`const`

Putting `const` in front of a variable just tells the compiler to allow the value of the variable to be changed.

```
const int DoNotChangeMe = 100;
```

```
DoNotChangeMe = 50;
```

```
constDemo.c: In function ‘main’:
constDemo.c:7: error: assignment of read-only variable
‘DoNotChangeMe’
```

Sample Final Exam

Which of the following is the correct two's complement representation for the value -135 if represented in 8 bits?

```
135

      1     2     4     8     16    33    67    135
      1     0     0     0     0     1     1     1

      0     1     1     1     1     0     0     0
+                                             1
==============================================
      0     1     1     1     1     0     0     1
```

01111001_2

Sample Final Exam

For a binary floating point number 10.1, which of the following would be true about its representation?

10.1_2

0.101×2^2

Sign

    0 (because it is positive)

Exponent in binary

    10_2 (2_{10})

Mantissa

    101_2 (everything to the right of the decimal)

Display a Linked List Using Recursion

So what would we pass to the recursive function?

The head of the linked list

So what condition would cause the recursive function to stop recursing and return?

The current node being NULL

What would we pass to the function in the recursive call?

The current node's `next_ptr`

Display a Linked List Using Recursion

```
void DisplayLinkedListRecursively(node *LLNode)
{
     if (LLNode == NULL)
     {
          return;
     }
     else
     {
          printf("\nNode Number %d\n", LLNode->node_number);

          DisplayLinkedListRecursively(LLNode->next_ptr);
     }
}
```

Sample Final Exam

Write the code that is needed to define the struct type *SHMData* needed for the program. The struct would be defined at the comment near the top of the program (at lines 7 – 10). Write the struct definition on this page. Use typedef to name the struct type.

```
typedef struct
{
      char *name;
      int iVal;
      char *cpItem;
      char cLtr;
      float fNum;
} SHMData;
```

Sample Final Exam

Given the input data and the structure of the print method (line 57), write the first two lines of data that would be printed by the print method if the method were completed.

```
Val         Name    Num Item                 Ltr

1941 Wonder_Woman 175.20 DC                     F
1962       Hulk 244.80 Marvel                 M
2004   Aquagirl 173.90 DC                     F
1985 Doctor_Light 159.10 DC                     F
1980     Cyborg 198.40 DC                     M
1962    Ant-Man 211.30 Marvel                 M
```

Sample Final Exam

In the print method, complete the lines of code (lines 72 – 73) needed to print every struct following the format given by the column headers above.

```
printf("\n %4d %10s %6.2f %-17s %4c",
          many[cnt].iVal, many[cnt].name,
          many[cnt].fNum, many[cnt].cpItem,
          many[cnt].cLtr);
```

Sample Final Exam

Write the lines that would assign the token *tok* to the *name* element of the struct at lines 30 and 31, Write the code below this question not on the program page.

```
many[cnt].name = malloc(sizeof(*tok)*sizeof(char));
strcpy(many[cnt].name, tok);
```

Sample Final Exam

Write the assignment to continue to use the tokenizer on the same input line (at line 36).

```
tok = strtok(NULL, " ");  // continue using the tokenizer
```

Sample Final Exam

Write the assignment that would take the integer value from the tokenizer and assign it to the integer element of the struct (at line 39).

```
many[cnt].iVal = atoi(tok); // convert and store
```

Sample Final Exam

Write the remaining lines of code needed to read the other values from the data file and store in the struct in the array (lines 45 – 48).

```
if (tok != NULL)
{
          many[cnt].cpItem = malloc(sizeof(*tok)*sizeof(char)); // make space
          strcpy(many[cnt].cpItem, tok);

          tok = strtok(NULL, " ");
}

if (tok != NULL)
{
          many[cnt].cLtr = *tok;

          tok = strtok(NULL, " ");
}

if (tok != NULL)
{
          many[cnt].fNum = atof(tok);
}
```

Sample Final Exam

Assume there is a struct like the one defined for the previous problem (*`SHMData`*) which now also includes a pointer to the same struct type. Also assume that the structure of the code as used before will be used again except that instead of storing the data elements into `many[cnt]`, you should assume that there is a temporary *`SHMData`* struct (not a pointer to a struct) called *`currentSHM`* that is being filled with the data from one line in the data file *`inFile`* in each iteration of the while loop.

```
SHMData currentSHM;
```

Sample Final Exam

Define a pointer called *head* that can point to an `SHMData` struct.

```
SHMData *head = NULL;
```

Sample Final Exam

Declare additional *SHMData* pointers as needed to be able to create a linked list of *SHMData* structs that will have *head* as the first element in the linked list.

```
SHMData *TempPtr = NULL;
SHMData *NewNode = NULL;
```

Sample Final Exam

Write the needed lines to store the filled struct *currentSHM* in the appropriate pointer from the previous question and link it into the linked list. Make sure to copy *currentSHM* not just point to it. Assume that this code will sit inside the while loop that reads the file but after all of the data elements for one line are read and stored into *currentSHM*.

```
currentSHM.name = malloc(sizeof(*tok)*sizeof(char)); // make space
strcpy(currentSHM.name, tok); // & store name
```

```
currentSHM.iVal = atoi(tok);
```

```
currentSHM.cpItem = malloc(sizeof(*tok)*sizeof(char));
strcpy(currentSHM.cpItem, tok);
```

```
currentSHM.cLtr = *tok;
```

```
currentSHM.fNum = atof(tok);
```

```
NewNode = malloc(sizeof(SHMData));
NewNode->next_ptr = NULL;
NewNode->name = currentSHM.name;
NewNode->iVal = currentSHM.iVal;
NewNode->cpItem = currentSHM.cp;
NewNode->cLtr = currentSHM.c;
NewNode->fNum = currentSHM.f;

if (head == NULL)
{
      head = NewNode;
}
else
{
      TempPtr = head;
      while (TempPtr->next_ptr != NULL)
      {
            TempPtr = TempPtr->next_ptr;
      }
      TempPtr->next_ptr = NewNode;
}
```

Sample Final Exam

Write a print method that will take in *head* and RECURSIVELY print all the data in all the elements of the linked list.

```
void print_LL(SHMData *node)
{
      if (node == NULL)
            return;
      else
      {
            printf("\n %4d %10s %6.2f %-17s %4c",
            node->iVal, node->name,
            node->f, node->cp, node->c);

            print_LL(node->next_ptr);
      }
}
```