



Final Exam Review

Review : CS60 . 141

Winter 2018 - University of Windsor
Dr. Sherif Saad



Agenda

Recursion

Pointers

Strings

File I/O

Recursion: Notes

- Any recursive function consist of at least one base case and one recursive case.
- The base case is when the recursive function encounters a sub-problem that it can solve without calling itself.
- The recursive case occurred when the recursive function encounters a sub-problem that could only be solved by calling itself with a smaller version of the sub-problem.

Recursion: Notes

```
IF (base case exist)
    Return some base case value
Else If (another case base exist)
    Return some other base case value
Else
    // do some work
    Return (recursive call)
```

Recursion - Exercise 01

Write a recursion function to calculate the factorial.

$$n! = 1 \quad \text{If } n = 0$$

$$n! = n * (n - 1)! \quad \text{If } n > 0$$

Recursion - Exercise 01

```
#include<stdio.h>

int factorial(int n){
    if (n == 0){ // base case when n = 0
        return 1;
    }
    else if( n== 1){ // base case when n = 1
        return 1;
    }
    return n * factorial(n-1); // recursive case
}

int main(){
    int value = 0;
    printf("please enter a value for n:\n");
    scanf("%d", &value);
    printf("factorial of %d is %d\n", value, factorial(value));
}
```

Recursion - Exercise 02

Write a recursive function to print all the elements in an array of integers of size n

Recursion - Exercise 02

```
#include <stdio.h>

int size;

void printArray(int array[], int index){

    if(index == size){
        return;
    }
    else{
        printf("item @ [%d] = %d\n", index, array[index]);
        index++;
        return printArray(array, index);
    }

}

int main()
{
    int vals[] = {3, 5, 8, 9, 0, 2, 6};
    size= sizeof(vals)/sizeof(int);
    printArray(vals, 0);
    return 0;
}
```


Recursion - Exercise 03

Write a recursive function to calculate the sum of all the elements in an array of integers of size n

Recursion - Exercise 03

```
#include <stdio.h>

int size;

int sumArray(int array[], int index){

    if(index == size){
        return 0;
    }
    else{
        return array[index++] + sumArray(array, index);
    }

}

int main()
{
    int vals[] = {3, 5, 8, 9, 0, 2, 6};

    size= sizeof(vals)/sizeof(int);

    int sum = sumArray(vals,0);
    printf("sum = %d", sum);
    return 0;
}
```

Pointers in C

- Pointers are **special variables** that store (points to) memory **addresses**
- Pointer **only** points to memory addresses.
- Pointers are a powerful tool for programmers and developers.
- Pointers could be **modified arithmetically** to point to any arbitrary address.
- In C/C++ pointers are **first-class (citizen) language** concept. Unlike python, java, and other languages.
- C#, Golang pointers are first-class (citizen).

```

#include <stdio.h>
int *ptr = NULL;

int fun_03(int *val, char c){
    printf("fun_03 says  %d, %c\n", *val, c);
    return ++(*val);
}

void fun_02(int *val){
    if(*val % 3 == 0){
        *val = fun_03(val, 'a');
    }
    else{
        *val = 10;
    }
    printf("fun_02 says value of val = %d\n", *val);
}

int fun_01(int val1, int val2){
    fun_02(&val1);
    printf("fun_01 says value of val1 = %d\n", val1);
    *ptr = val1 * fun_03(&val2, 'b');
    printf("fun_01 says value of ptr = %d\n", *ptr);
    return 0;
}

```

Exercise 04

Trace and show the output

```

int main()
{
    int a = 15;
    int b = 7;
    int c;

    ptr = &c;

    fun_01(a, b);

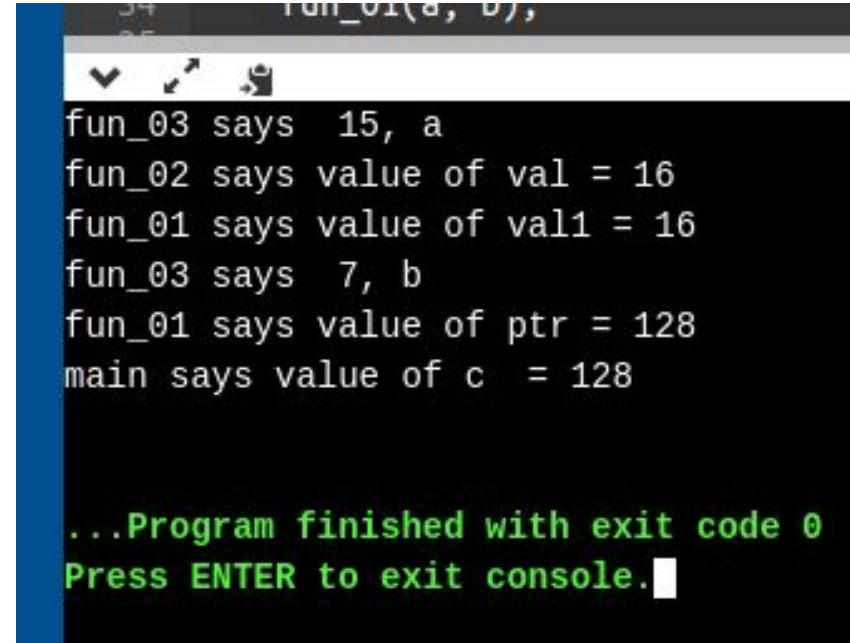
    printf("main says value of c  = %d\n", c);
    return 0;
}

```

Exercise 04

When you trace a code always try to show your work

Writing the final output is only good if it is correct.

A screenshot of a console window with a black background and white text. The output shows a sequence of messages from different functions: 'fun_03 says 15, a', 'fun_02 says value of val = 16', 'fun_01 says value of val1 = 16', 'fun_03 says 7, b', 'fun_01 says value of ptr = 128', and 'main says value of c = 128'. At the bottom, green text indicates the program finished with exit code 0 and prompts the user to press ENTER to exit the console. The console window has a blue title bar and standard Windows window controls (minimize, maximize, close) are visible at the top.

```
fun_03 says 15, a  
fun_02 says value of val = 16  
fun_01 says value of val1 = 16  
fun_03 says 7, b  
fun_01 says value of ptr = 128  
main says value of c = 128  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Exercise 05

Which print statement will print the value of var1?

```
int main()
{

    int  var1 = 7;
    int *ptrInt1, *ptrInt2;

    ptrInt1= &var1;
    ptrInt2 = NULL;

    printf("%u: value of ptrInt1\n",ptrInt1);
    printf("%p: value of ptrInt1\n",ptrInt1);
    |
    printf("%u:  value of ptrInt2\n",ptrInt2);
    printf("%p:  value of ptrInt2\n",ptrInt2);

    return 0;
}
```

Exercise 05

Which print statement will print the value of var1?

None

```
1078239756: value of ptrInt1
0x7ffd4044a20c: value of ptrInt1
0: value of ptrInt2
(nil): value of ptrInt2

...Program finished with exit code 0
Press ENTER to exit console.
```

```
int main()
{

    int var1 = 7;
    int *ptrInt1, *ptrInt2;

    ptrInt1= &var1;
    ptrInt2 = NULL;

    printf("%u: value of ptrInt1\n",ptrInt1);
    printf("%p: value of ptrInt1\n",ptrInt1);
    |
    printf("%u: value of ptrInt2\n",ptrInt2);
    printf("%p: value of ptrInt2\n",ptrInt2);

    return 0;
}
```

Common Mistakes 1

In the code on the right side. The pointer **ptrInt** is pointing to the address **33** in memory and **not to** the address of **var1**



```
int main()
{

    int var1 = 33;

    int *ptrInt;

    ptrInt = var1;

    printf("%u\n", ptrInt);

    printf("%p\n", ptrInt);

    return 0;

}
```


Common Mistakes 2

In this case, you are not incrementing the value of var1 using the pointer. The value of var1 still equals 33.

What you did has you changed the address that the ptrInt was pointing to. Now ptrInt is not pointing to var1

```
int main()
{
    int var1 = 33;

    int *ptrInt;

    ptrInt = &var1;

    ptrInt++;

    printf("%u\n", ptrInt);

    printf("%p\n", ptrInt);

    return 0;
}
```

Common Mistakes 3

Here we use a pointer to type int to points to a variable of type float.

Note the code will compile but it could have unexpected behaviors

```
int main()
{

    float var1 = 3.14;

    int *ptrInt;

    ptrInt = &var1;

    printf("%u\n",ptrInt);
    printf("%p\n",ptrInt);

    return 0;
}
```

Pointer and Strings

- A string is a sequence of characters in which the last character has a numerical value of 0 or null character.
- The value of the null character is represented by the escape sequence '\0'. This value indicates the end of the string
- A string variable is an array of type char.

```
char str[20]; //string variable; can hold up to 20 characters
```

String Constant

An example of a string constant is:

```
char str1 [6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
char str2 [6] = "Hello";
```

```
char str3 [] = "Hello";
```

Note that a string literal must end with the null character the compiler will add the null character to the string. This is why the word **Hello** is stored in an array of 6 characters.

The `sizeof` Operator

- The `sizeof` operator queries size of the object or type.
- We use it when actual size of the object must be known.
- Returns size in bytes of the variable of any type.

```
int var = 250;  
printf("%d bytes size of var", sizeof(var));
```

```
4 bytes size of var
```

```
...Program finished with exit code 0  
Press ENTER to exit console. █
```

String Library: `string.h`

- The **string.h** library contains useful functions to copy strings, compare strings, search for a character or sub-string in another string, and so on.
- The most common functions in string.h are:
 - **strlen(str)**: return the length (number of non null characters) of the string str
 - **strcpy(dest, src)**: copies the contents of src into dest
 - **strcat(str1, str2)**: append str2 to str1
 - **strcmp(s1,s2)**: compare s1 and s2 if they are equal it return 0, if s1 greater than s2 it return 1 otherwise it return -1
 - **strstr(str, key)**: return the first index of the string in key if it is a substring of str

Exercise 06

Write a function that count the occurrence of a specific character in a given string.

Example in String = "Hello World!" the character 'l' occurred 3 times

Exercise 06

The function has two arguments one to store the character we want to count its occurrence and the second store the string we want to search.

```
int countChar(char key, char *  
str);
```

```
#include <stdio.h>

int countChar(char key, char * str){
    int count = 0;

    while(*str){
        if (*str++ == key){
            count++;
        }
    }
    return count;
}

int main(){
    char *str = "Hello worlds!";
    int c = countChar('l', str);
    printf("%d occurrences",c);
    return 0;
}
```


Exercise 07

Write a function `copystr(char * dest, char * src)` that copy the contents of one string (`src`) into another string (`dest`)

Exercise 07

```
#include <stdio.h>
void copystr(char *dest, char *src){

    while(*src){
        *dest++ = *src++;
    }
    *dest = '\0';
}
```

Exercise 07

```
int main(){

    char str1[] = "Hello World";
    char str2[25];

    copystr(str2, str1);

    printf("%s", str2);
    return 0;

}
```

```
#include <stdio.h>

void copystr(char *dest, char *src){

    while(*src){
        *dest++ = *src++;
    }
    *dest = '\0';
}

int main(){

    char str1[] = "Hello World";
    char str2[25];

    copystr(str2, str1);

    printf("%s", str2);
    return 0;

}
```

Exercise 08

Write a function `CopyAlpha(char * dest, char * src)` that copy only the alphabetic characters of one string (`src`) into another string (`dest`)

Exercise 08

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

void CopyAlpha(char *str1, char *str2){

    while(*str1){

        if(isalpha(*str1)){
            *str2++ = *str1++;
        }else{
            *str1++;
        }
    }
    *str2 = '\0';
}
```

Exercise 09

Explain the function codeX

```
char * codeX(char * str, int start, int end){  
  
    if(end < start || end > sizeof(str)){  
        return str;  
    }  
  
    char * output = (char*)malloc(sizeof(end-start));  
  
    for(int i= 0; i<sizeof(output); i++){  
        *(output+i) = *(str + (start+i));  
    }  
    return output;  
}  
  
int main()  
{  
    char text[] = "This_is_the_CS_141_Final_Exam";  
    char * ptr;  
  
    ptr = codeX(text, 0, 9);  
    printf("codeX = %s\n", ptr);  
}
```

Creating Structures in C

The keyword **struct** introduces the structure definition using struct statement.

The struct statement defines a new data type, with more than one member. The format of the struct statement is

```
struct <structure name> {  
    member definition;  
    ...  
    member definition;  
} one or more structure variables;
```

Creating Structure Example

The following struct statement defines book structure as a collection of title, author, subject and book ID. Then it declares two variables b1 and b2 of type Book

```
struct Book {  
    char title[50];  
    char author[50];  
    char subject[100];  
    int book_id;  
  
} b1, b2 ;
```


Creating Structure Example

After creating the structure of type X (e.g., Book) using the struct statement, we can declare a new variable of type X in place in our C program.

```
struct Book {  
    char title[50];  
    char author[50];  
    char subject[100];  
    int book_id;  
};  
  
int main(){  
    struct Book myBook;  
}
```

Notes on Creating Structure

- Variables declared within the braces of the structure definition are the **structure's members**.
- Members of the same structure type must have unique names, but two different structure types may contain members of the same name without conflict.
- Each structure definition must end with a **semicolon**.
- Structure members can be variables of the **primitive data types** or variables of **other structures**.

Creating Structure Example

The author of a Book structure is a variable of type Author, which is another structure.

```
struct Author {  
    char firstName[25];  
    char lastName [25];  
};  
struct Book {  
    char title[50];  
    Author author;  
    char subject[100];  
    int book_id;  
  
};  
int main(){  
    struct Book myBook;  
}
```

Notes on Creating Structure

Structure members **cannot** be initialized with a declaration.

```
struct Point2D {  
    int x = 0;  
    int y = 0;  
};
```

This because when a datatype is declared, no memory is allocated for it. Memory is allocated only when variables are created.

Notes on Creating Structure

A structure **cannot** contain an **instance of itself**.

```
struct Point2D {  
    int x = 0;  
    int y = 0;  
    struct Point2D p1;  
};
```

Struct Point2D contains an instance of itself (p1), which is an error.

Notes on Creating Structure

A structure definition **may** contains a **pointer to itself**

```
struct Node{  
    int node_id;  
    float value;  
    char label[5];  
    struct Node * ptrNode;  
};
```

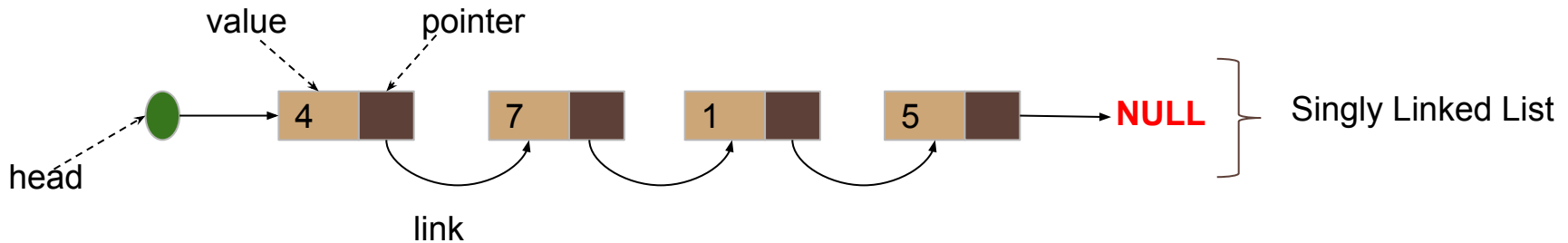
A structure containing a member that's a pointer to the same structure type is referred to as a **self-referential structure**.

Self-referential structures are used to create more complex and efficient data structure

Linked List

What is a Linked List?

- Elements are located in a linear order.
- A collection of nodes where each node has a value and points to the next node in the list.



Add/Insert new Node

Here the function will always insert the new node at the beginning of the list

```
int main() {  
    ListNode *head;  
  
    InitiateList(&head);  
  
    InsertAtBeginning(4, &head);  
}
```

```
void InsertAtBeginning(int v, ListNode **head){  
  
    ListNode *node = (ListNode*)malloc(sizeof(ListNode));  
  
    node->value = v;  
    node->next = NULL;  
  
    if (*head==NULL){  
        *head = node;  
    }else{  
        node->next = *head;  
        *head = node;  
    }  
}
```


Add/Insert new Node

Here the function will always insert the new node at the end of the list

```
int main() {  
  
    ListNode *head;  
  
    InitiateList(&head);  
  
    InsertAtBeginning(4, &head);  
    InsertAtBeginning(7, &head);  
  
    InsertAtEnd(9, &head);  
}
```

```
void InsertAtEnd(int v, ListNode **head){  
  
    ListNode *node = (ListNode*)malloc(sizeof(ListNode));  
  
    node->value = v;  
    node->next = NULL;  
  
    ListNode * current = *head;  
  
    while(current->next!=NULL){  
  
        current = current->next;  
    }  
    current->next = node;  
}
```

Delete a node

This function will delete the first node in the list. What if the list is empty?

```
int main() {  
  
    ListNode *head;  
  
    InitiateList(&head);  
  
    InsertAtBeginning(4, &head);  
    InsertAtBeginning(7, &head);  
  
    InsertAtEnd(9, &head);  
  
    printf("List Size = %d\n", ListSize(head));  
  
    DeleteAtBeginning(&head);  
}
```

```
void DeleteAtBeginning(ListNode **head){  
  
    ListNode *node = *head;  
  
    *head = (*head)->next;  
  
    free(node);  
}
```

Delete a node

This function will always remove the last node in the list. What if the list is empty?

```
int main() {  
  
    ListNode *head;  
  
    InitiateList(&head);  
  
    InsertAtBeginning(4, &head);  
    InsertAtBeginning(7, &head);  
  
    InsertAtEnd(9, &head);  
  
    printf("List Size = %d\n", ListSize(head));  
  
    DeleteAtBeginning(&head);  
  
    DeleteAtEnd(&head);  
  
}
```

```
void DeleteAtEnd(ListNode **head){  
  
    ListNode *current = *head;  
  
    while(current->next->next!=NULL){  
        current = current->next;  
    }  
  
    ListNode *node = current->next;  
    current->next = NULL;  
    free(node);  
  
}
```

Find the Size of the Linked List

The function will return the total number of nodes in the list.

```
int main() {  
  
    ListNode *head;  
  
    InitiateList(&head);  
  
    InsertAtBeginning(4, &head);  
    InsertAtBeginning(7, &head);  
  
    InsertAtEnd(9, &head);  
  
    printf("List Size = %d\n", ListSize(head));  
  
}
```

```
int ListSize(ListNode *head){  
    int size = 0;  
  
    ListNode *current;  
    current = head;  
  
    while(current!=NULL){  
        size = size + 1;  
        current = current->next;  
    }  
  
    return size;  
}
```

Searching a Linked List

```
void SearchList(int v, ListNode *head) {  
  
    ListNode *current;  
    current = head;  
    int index = -1;  
    int count = 0;  
    while (current != NULL) {  
  
        if(current->value == v){  
            index = count;  
            return index;  
        }  
  
        current = current->next;  
        count += 1;  
    }  
    return index;  
}
```

Clear All Nodes in the List

Here the function will delete all the nodes in the list

```
void ClearList(ListNode **head){  
  
    ListNode *current = *head;  
  
    while(current!=NULL){  
        current = current->next;  
        free(*head);  
        *head = current;  
    }  
}
```

Sorting a Linked List

A linked list is a linear data structure similar to array but it is dynamic. We could use any standard sort algorithm like bubble sort, insertion sort or selection sort.

Let us use selection sort to sort the data or the value in the list.

Note when we need to swap we only swap the data, we do not swap the pointer or the node

Sorting a Linked List

```
void SortList(ListNode *head){
    ListNode *current;
    ListNode *after;
    int temp;

    for(current = head; current->next!=NULL; current = current->next){

        for(after =current->next; after!=NULL; after=after->next){

            if(current->value > after->value){
                temp = current->value;
                current->value = after->value;
                after->value = temp;
            }
        }
    }
}
```


Exercise 10

Write a function called `PrintList` that print all the items in a linked list recursively

Exercise 11

Write a function called `PrintList` that count all the vowel letters in a linked list of characters recursively

Exercise 10

Write a C program that read the records of n number of students from the user input. Each student record consist of the student last name, number of courses the student enrolled in and the student GPA. The program should sort the student list based on their GPA in descending order and store all the records in a file binary or text file.

Exercise 12

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

typedef struct student{
    char lname[25];
    int courses;
    float gpa;
}Student;
```

Exercise 12

```
int main() {
    printf("This program demo basic File I/O operations\n");

    int num;
    printf("enter the number of students records you want to store: ");
    scanf("%d",&num);

    Student * list = (Student *)malloc(sizeof(Student)*num);

    FILE *fptr = NULL;
    fptr = fopen("list.txt", "w");

    if(fptr == NULL){
        printf("Error can not write to a file");
        exit(1);
    }
}
```

Exercise 12

```
for(int i=0; i < num; i++){  
  
    printf("enter student last name: ");  
    scanf("%s", (list+i)->lname);  
  
    printf("enter number of courses: ");  
    scanf("%d", &(list+i)->courses);  
  
    printf("enter student GPA: ");  
    scanf("%f", &(list+i)->gpa);  
}
```

Exercise 12

```
for(int i =0; i<num-1; i++){  
    for(int j =i+1; j<num; j++){  
        if((list+i)->gpa<(list+j)->gpa){
```

```
            Student temp;
```

```
            temp.gpa = (list+i)->gpa;  
            temp.courses = (list+i)->courses;  
            strcpy(temp.lname,(list+i)->lname);
```

```
            (list+i)->gpa = (list+j)->gpa;  
            (list+i)->courses = (list+j)->courses;  
            strcpy((list+i)->lname,(list+j)->lname );
```

```
            (list+j)->gpa = temp.gpa;  
            (list+j)->courses = temp.courses;  
            strcpy((list+j)->lname,temp.lname );
```

```
        }
```

```
    }
```

```
}
```

Exercise 12

```
for(int i=0; i < num; i++){  
  
    fprintf(fp_ptr, "%s\n", (list+i)->lname);  
    fprintf(fp_ptr, "%d\n", (list+i)->courses);  
    fprintf(fp_ptr, "%f\n", (list+i)->gpa);  
    fprintf(fp_ptr, "\n-----");  
}  
  
fclose(fp_ptr);  
return 0;  
}
```


Questions