**CSCD 240-Fall 2020 Name: Ian Kaiserman**

**Quiz: 4 (Struct and Linked List) Date:** **11/30/20**

1. (a) Define a structure of type **complex** consisting of two floating-point members, called **real** and **imaginary**. 1 pt

**Struct complex {**

**Float real;**

**Float imaginary;**

**}**

(b) How will you find the size of the **struct** as mentioned in (a)? 1 pt

**Sizeof(struct complex)**

(c) Declare a variable x that is a structure of type complex. Initialize each member of the variable with the following values: 2 pts

real: 24.5

imaginary: 34.5;

**struct complex x = {24.5, 34.5};**

(d) Declare a one-dimensional 1000-element array called **complexArray** whose elements are structures of type complex. Write expression for the members of the 100th array element (i.e., element number 99); 4 pts (2 + 2)

**Struct complex complexArray[1000];**

**complexArray[99].real = …**

**complexArray[99].imaginary = …**

1. What is ‘typedef’? Declare a user-defined data type **complex\_number** within the definition of complex. 2 pts

**Typedef is a way of taking a long declaration of an object or type and making an alias for it that can be reused later.**

**Typedef complex\_number = struct complex {**

**Float real;**

**Float imaginary;**

**};**

3. Define a self-referential structure **book\_list** with the following members:

(a) a character array called **title** of length 40; 1 pt

(b) a sub-structure with variable name **data** with members (i) **isbn** of type 20-element character array and (ii) **price** of type double;

2 pts

(c) a pointer to another structure of the same type i.e., **book\_list**, called **next**. 1 pt

Struct book\_list {

Char title[40];

Struct data {

Char isbn[20];

Double price;

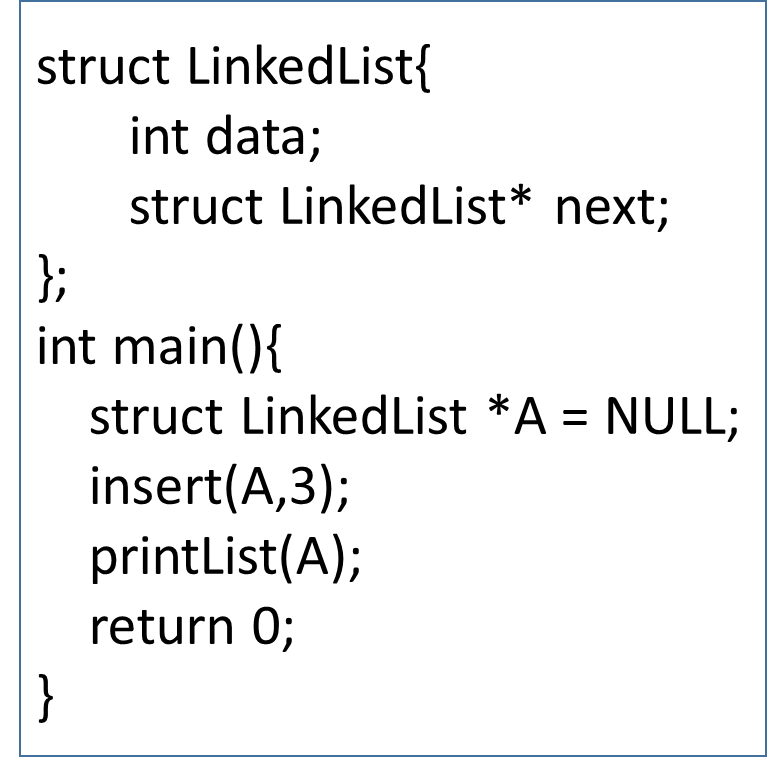
};

Struct book\_list \*next;

}

4. Is there any problem in the following program? How will you fix it?

2 pts (1 + 1)



**void insert(struct LinkedList \*root, int item){**

**root=malloc(sizeof(struct LinkedList));**

**root->data= item;**

**root->next = NULL;**

**}**

**void printList(struct LinkedList \*head){**

**struct LinkedList \*temp=head;**

**while (temp != NULL){**

**printf("%d", temp->data);**

**temp = temp->next;**

**}**

}

**The insert function is unable to modify the existing data and attributes of the head node the way it’s set up. The function needs to be altered so that it takes a pointer to a LinkedList pointer (\*\*root), create a new LinkedList pointer with the new item int, and reassign \*root to the new LinkedList pointer. Because of this, the function call should be printList(&A) instead.**

Consider the following program to answer Question 5 and Question 6. Assume the executable version of the code is named **myprog**.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char \*\*argv){

struct node{

int number;

struct node\* next;

};

struct node\* head = NULL;

struct node\* temp;

int i;

if (argc < 2) exit(1);

int num = atoi(argv[1]);// 'atoi' takes string as argument, and converts to the equivalent int value and

// returns it.

printf("num is %d\n", num);

for (i = 1; i <= num; i++){

if (head == NULL){

head = malloc(sizeof( struct node));

temp = head;

}

else{

temp->next = malloc(sizeof(struct node));

temp = temp->next;

}

temp->number = i;

}

temp = head;

while (temp != NULL){

printf("%d\n", temp->number);

temp = temp->next;

}

return 0;

}

5. What will be printed by the program if it is invoked with a command line argument as mentioned below?

**./myprog 4 3 pts**

**num is 4**

**1**

**2**

**3**

**4**

6. Does the program insert nodes at the beginning of the list or at the end? What does the variable ‘**number**’ in struct node do in the program? **2 pts (1 + 1)**

**Nodes are inserted at the end. Temp is sent to the end of the list and allocates memory for the next reference of the last element.**

**‘number’ is storing the integer value of the struct node that is being assigned by the loop and printed at the end.**

**Identify whether the following statements are True or False: 4 pts**

(1) Inserting an element at any point in the middle of an array is easier than inserting an element at any point in the middle of a linked list.

**false**

(2) Consider the following program:

struct complex{

double real;

double imaginary;

}x;

struct complex \*ptr = &x;

x.real = 23.0;

x.imaginary = 45.12;

The following two expressions are equivalent:

(&x)->real and ptr->real

**true**

(3) The following expression is correct.

struct temp{

float x;

float y;

struct temp z;

};

**false**

(4) Consider the following code snippet. The program will insert two nodes in the linked list.

void push(struct node\* head, int data){

struct node{

int data;

struct node\* next;

};

struct node\* newnode = malloc(sizeof(struct node));

newnode->data=data;

newnode->next= head;

head = newnode;

}

int main(){

struct node\* A = NULL;

push(A, 5);

push(A, 6);

return 0;

}

**false**