

ECE220 Final Review - Cramming Carnival

Author: Members of HKN

Fall 2024

1 Logical Little Computer 3 Language Lessons

- (a) **I/O:** What is the difference between Polling I/O versus Interrupt I/O?

- (b) **Stack:** When using a stack, when you pop an item off the stack, is it removed from memory?

- (c) **Stack Pt. 2:** If the input stream to a Stack is 123456789, design a set of pushes and Pops such that it is outputted 235641897?

- (d) **Subroutines:** What is the purpose of using subroutines in LC3?

- (e) **Multiplication:** Write a Multiplication Subroutine in LC3 assembly language.
- ```
;input R3, R4
;out R0
MULTIPLY
;your code goes here:
```

- (f) **More subroutines:** What extra step do you need to take when executing a subroutine inside a subroutine? (Hint: What gets overwritten when JSR is called?)

## 2 Captivating C Coding

(a) **Variable lifespan:** Explain the “lifespan” of a local variable during a C function call.

(b) **Poopy Pointers:** Explain the importance of having the parameters of this function being pointers.

```
1 void swap(int* a, int* b)
2 {
3 int temp;
4 temp = *a;
5 *a = *b;
6 *b = temp;
7 }
```

(c) **Lovely Linked Lists:** What is the benefit of using a linked list over an array to represent a large, ever-changing list?

(d) **Rambunctious Recursion:** When writing a recursive algorithm, what is the goal of each recursive step? (Hint: The base case represents the simplest form of the problem)

(e) **Arrays:** How are arrays passed into functions in C?

(f) **Static Silliness:** What is printed by the following program?

```
1 static char letters[6] = {'A', 'E', 'F', 'D', 'B', 'C'};
2
3 void mystery () {
4 static int X = 5;
5 int Y = 2;
6 printf ("%c%c", letters[--Y], letters[X--]);
7 }
8 int main() {
9 mystery();
10 mystery();
11 return 0;
12 }
13
```

(g) **Midpoint:** What is wrong with this recursive midpoint function?

```
1 find_midpoint(int a, int b) {
2 if (a == b) { return a; }
3 else if (a < b){
4 return find_midpoint (a+1, b-1);
5 }
6 else if (a > b){
7 return find_midpoint(a-1, b+1);
8 }
9 }
```

- (h) **Function Foo:** What is this function accomplishing? Assume it is called with a valid head to a singly linked list.

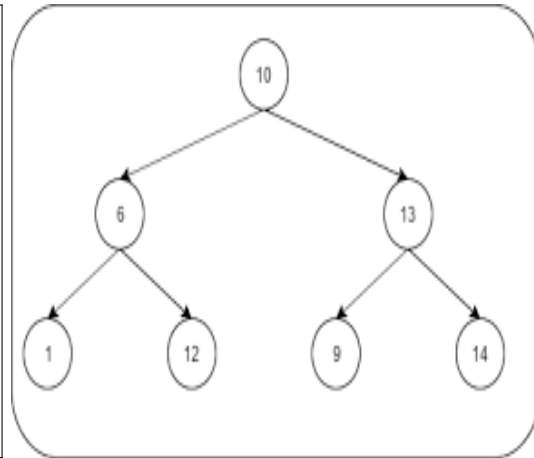
```
1 struct node{
2 int num;
3 struct node* next;
4 }
5 int foo(struct node* head, int bar) {
6 if (head->num >= bar) {
7 bar = head->num;
8 }
9 if (head->next == null) {
10 return bar;
11 }
12 else {
13 return foo(head->next, bar);
14 }
15 }
```

- (i) Which sorting algorithm is this?

```
1 void sort(int arr[], int n) {
2 int i, j;
3 for (i = 0; i < n - 1; i++)
4 for (j = 0; j < n - i - 1; j++)
5 if (arr[j] > arr[j+1])
6 swap(&arr[j], &arr[j + 1]);
7 }
```

(j) **Binary Search Trees:** Are these binary search trees?

figures/bst.png



(k) **C to LC3:** Convert the following C function into LC3 using Callee Setup and Teardown.

```
1 int foo(int a, int b) {
2 int x;
3 x = a + b;
4 return x;
5 }
```

- (l) **Recursive Reversal:** What is the output of this program? If there is an error in ReverseArray, identify the line and fix it? (Hint: it might be nice and helpful to print every step of Reverse Array)

```
1 void ReverseArray(int array[], int size) {
2 int start = 0;
3 int end = size - 1;
4 int temp;
5
6 if (start < end) {
7 // Swap First and Last
8 temp = array[start];
9 array[start] = array[end];
10 array[end] = temp;
11
12 ReverseArray(array, size - 1);
13 }
14 }
15 int main(){
16 int array[5], i;
17 for (i = 0; i < 5; i++){
18 array[i] = i;
19 }
20 ReverseArray(array, 5);
21 printf("Reversed Array: ");
22
23 for (i = 0; i < 5; i++){
24 printf("%d ", array[i]);
25 }
26 printf("\n");
27 return 0;
28 }
```

- (m) What is wrong with the following code?

```
1 double * firstlast(const double values[], int size)
2 {
3 double result[2];
4 result[0] = value[0];
5 result[1] = value[size - 1];
6 return result;
7 }
```

- (n) **Alternative Indexing:** Fill in the blanks to make the two arrays the same. Assume row-major order



for array2.

```
1 int array1[4][2];
2 int array2[8];
3 int i, j;
4 for (i = 0; i < 2; i++) {
5 for (j = 0; j < 4; j++) {
6 array1[i][j] = i + j;
7 array2[i+j] = i + j;
8 }
9 }
```

- (o) Fill In the blanks to find the student with the highest GPA and store a pointer to them in best\_student

```
1 typedef struct StudentStruct {
2 int UIN;
3 float GPA;
4 } Student;
5
6 int main () {
7 Student all_students[5];
8 // Load data into all students:
9 load_students(all_students, 5);
10 // Find the student with the highest GPA:
11 Student* best_student =
12 find_best(all_students, 5,
13);
14 printf("Best GPA:%f\n",
15);
16
17 void find_best(Student* all, int num_students, Student** best) {
18 for (int i = 0; i < num_students; i++) {
19 if (all[i].GPA >
20) {
21 // Fill:
22 }
23 }
24 }
```

- (p) In MP10, we tasked you with inserting a node into a sorted singly-linked list. To help you get accustomed to poopy pointer arithmetic, we will now attempt to do the same, but with a doubly-linked list. You are given the following struct doubly\_node and the function insert\_doubly\_node(), which takes in the head of a doubly linked list (but you will not have a tail) and a value to insert. Complete the function skeleton below. Assume that the double-pointer head is never NULL, but that the single

pointer `*head` (the result of dereferencing `head` once) is not necessarily never NULL.

```

1 typedef struct doubly_node {
2 int value
3 doubly_node * next;
4 doubly_node * prev;
5 } doubly_node;
6
7
8 void print_list_d(doubly_node * head)
9 {
10 doubly_node * cur = head;
11 while (cur != NULL)
12 {
13 printf("%d -> ", cur->value);
14 cur = cur->next;
15 }
16 }
17
18 void insert_doubly_node(doubly_node ** head, int value)
19 {
20 /* edge case 0: linked list is completely empty */
21 if(*head ==)
22 {
23 *head = () malloc(sizeof());
24 (*head)->next = ;
25 (*head)->prev = ;
26 (*head)->value = ;
27 return;
28 }
29
30 /* edge case 1: insert AT the head*/
31 if (value < (*head)->)
32 {
33 doubly_node * new_node = () malloc();
34 new_node-> = value;
35 new_node->next = ; //new node should be inserted before head
36 new_node-> = NULL; //new node previous does not point to anything
37 (*head)-> = new_node; //head's previous points to new_node
38 *head = ; //update the new head of the linked list
39 return;
40 }
41
42 doubly_node * cur = *head;
43 while (cur !=)
44 {
45 if (cur->value < value)
46 {
47 /* last edge case: insert AFTER the tail of the linked list */
48 if (cur->next == NULL)
49 {
50 doubly_node * new_node = ;
51 //allocated new_node on the heap
52
53 new_node->value = value;
54 new_node->next = ; //new_node is the tail of the list
55 new_node-> = cur; //new_node's previous points to the current
56 cur->next = ; //cur's next is the new node
57 return;
58 }
59 cur = // advance to the next node
60 }
61
62
63 else
64 {
65 doubly_node * new_node = () malloc(sizeof());
66

```

```

67 new_node->value = ;
68 new_node->next = ;
69 new_node-> = cur->prev;
70 cur->prev->next = ;
71 cur->prev = new_node;
72 return;
73 }
74 }
75 }

```

Assuming that `insert_doubly_node()` and `print_list_d()` are implemented correctly, what will the output of this code snippet be?

```

1 int main()
2 {
3 doubly_node * head = NULL;
4 insert_doubly_node(&head, 5);
5 insert_doubly_node(&head, 2);
6 insert_doubly_node(&head, 7);
7 insert_doubly_node(&head, 9);
8 insert_doubly_node(&head, 1);
9 insert_doubly_node(&head, 10);
10 insert_doubly_node(&head, 5);
11 print_list_d(head);
12
13 return 0;
14 }

```

### 3 C++ Programming

- (a) **Class:** What is the difference between structs in C and classes in C++?
- (b) **Constructor:** What does a constructor do in a class? What does a destructor do in a class? What might happen if we do not have a correct destructor?
- (c) **Access specifier:** What is the difference between public, private, and protected in a C++ class?
- (d) **Operator overloading:** What does operator overloading do in a C++ class? Give four different examples of operators that can be overloaded.
- (e) **Rectangle:** Write the constructors and area function for this rectangle class.
- ```
1 class Rectangle {  
2     private:  
3         int width, height;  
4     public:  
5         Rectangle();
```

```

6     Rectangle(int w, int h);
7     int const area();
8 }

```

(f) **Iterator:** What will be printed if we run the following C++ code? (assume we import all needed files)

```

1     int main()
2 {
3     vector<int> ar = { 1, 2, 3, 4, 5 };
4     vector<int>::iterator ptr = ar.begin();
5     ptr++;
6     cout << *ptr << endl;
7     ptr = ptr + 3;
8     cout << *ptr << endl;
9     return 0;
10 }

```

(g) **Template:** What will happen if we run the main function?

```

1 template <typename T>
2 T max(T x, T y)
3 {
4     return (x > y)? x : y;
5 }
6 int main()
7 {
8     cout << max(3, 7) << endl;
9     cout << max(3.0, 7.0) << endl;
10    cout << max(3, 7.0) << endl;
11    return 0;
12 }

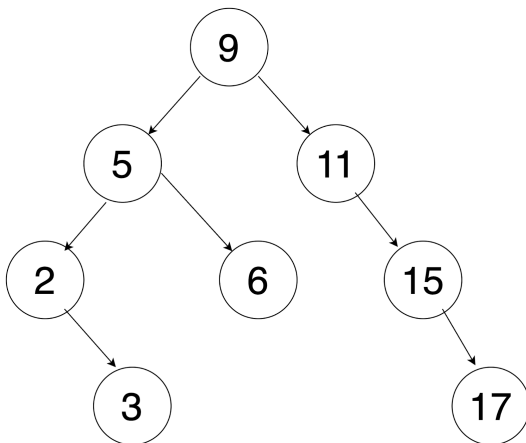
```

(h) **Copy:** What is the difference between shallow copy and deep copy?

(i) What type of traversal is this?

```
1 void traversal (node *root) {  
2     if ( root != NULL) {  
3         std::cout << root -> data << std::endl;  
4         traversal (root -> left);  
5         traversal (root -> right);  
6     }  
7 }  
8
```

(j) What is the pre-order, in-order, post-order and level order traversal for the following binary search tree?



- (k) How many bytes of memory have been allocated on the heap? Assume that we are in a 32-bit system (recall that an int is 4 bytes, a double is 8 bytes, an int * is 4 bytes, and a char is 1 byte)

```

1
2     int main () {
3         int a = 4;
4         int d = 6;
5         int c = 10;
6         int ** b = new int * [5];
7         b[0] = &a;
8         b[1] = &c;
9         b[2] = &d;
10        b[3] = new int [6];
11        b[3][0] = 5;
12        b[3][2] = 8;
13        b[3][1] = 17;
14        b[3][5] = 2;
15        b[3][4] = 9;
16        b[3][3] = 22;
17        b[4] = new int(13);
18        char * character = new char (k);
19    }
20
21
22

```

Write the contents of what each pointer points to (if it points to an array, write the contents of the array. If it points to a single int/char, write what the int/char is.)

Fill in the lines after line 10 needed to properly free the memory. Not all lines may be needed.

```

1     int main () {
2         int a = 4;
3         int d = 6;
4         int c = 10;
5         int ** b = new int * [5];
6         b[0] = &a;
7         b[1] = &c;
8         b[2] = &d;
9         b[3] = new int [6];
10        b[3][0] = 5;
11        b[3][2] = 8;
12        b[3][1] = 17;
13        b[3][5] = 2;
14        b[3][4] = 9;
15        b[3][3] = 22;
16        b[4] = new int(13);
17        char * character = new char ('k');
18        -----
19        -----
20        -----
21        -----
22        -----
23        -----
24    }
25
26

```

Let's say we have a void pointer p. What is the difference between delete [] p and delete p?

- (1) Linked List Problem: Your best friend Gabriel has a C program that stores the name of his favorite Fortune 500 company. He is storing the company name in a linked list format. The structure of the linked list and the nodes are located below. The end of the list will have the company _node * next point to NULL;

```
1      typedef struct company_node
2      {
3          int index;
4          char letter;
5          struct company_node * next;
6
7      } company_node;
8
9      typedef struct company
10     {
11         int n; //number of entries in list
12         company_node * company_head;
13
14     } company;
15
```

Given the above structure, and the fact that the company_node * pointer is 8 bytes (64-bit system), **what is the size in bytes of both struct company and struct company_node?**

Gabriel wants to tell you what his favorite Fortune 500 company is but is unable to speak because he just got his wisdom teeth removed. **Fill in the below program to print the name of the company.** This program takes place in main and has already imported the relevant libraries.

Assume everything has been initialized/malloc'd correctly and the name of the struct was declared with the following line.

Part 1: Printing

```
1      company * gabriel_company;
2
3      printf("The Name of Gabriel's Favorite Fortune 500 Company: ");
4
5      company_ * current = gabriel_company->company_head;
6      for (int i = 0; i < _____; i++) //iterate through the list
7      {
8          printf("%c", _____); //print the current character
9          _____ //move the current node
10     }
11
```

Gabriel is quite shy and doesn't want anyone to know what his favorite Fortune 500 company is (except for you, his best friend). **Help him finish the below code to remove the company name from the heap.**

Part 2: Freeing

```
1      company_node * the_current = gabriel_company->company_head;
2      company_node the_previous = gabriel_company->company_head;
3      while(the_current->next != _____){
4          the_current = the_current->next;
5          free(_____);
6          _____; //move previous further
7      }
8
9      free(gabriel_company);
10
```


Whenever the above code is executed, a memory leak occurs. /textbfExplain what a memory leak is and come up with one line of code that can be added above to fix the memory leak.

(m) Virtual Keywords:

```
1      #include <iostream>
2      class Base
3      {
4          public:
5              virtual void f()
6              {
7                  std::cout << "base\n";
8              }
9      };
10
11     class Derived_One : public Base
12     {
13     public:
14         void f() override
15         {
16             std::cout << "derived_1\n";
17         }
18     };
19
20     class Derived_Two : public Base
21     {
22     public:
23         void f()
24         {
25             std::cout << "derived_2\n";
26         }
27     };
28
29
30     int main()
31     {
32         //Define the objects
33         Base b;
34         Derived_One d_1;
35         Derived_Two d_2;
36
37         //assign the objects by reference
38         Base& br = b;
39         Base& dr_1 = d_1;
40         Base& dr_2 = d_2;
41
42         /* PART 1 */
43         br.f();
44         dr_1.f();
45         dr_2.f();
46
47         //Assign the objects by pointer
48         Base * bp = &b;
49         Base * dp_1 = &d_1;
50         Base * dp_2 = &d_2;
51     }
```

```

52         /* Part 2 */
53         bp->f();
54         dp_1->f();
55         dp_2->f();
56
57         /* Part 3 */
58         br.Base::f();
59         dr_1.Base::f();
60         dr_2.Base::f();
61
62         return 0;
63     }
64

```

Part 1: What is printed by br, dr_1, and dr_2?

Part 2: What is printed by br, dr_1, and dr_2? What is the difference between Part 1 and Part 2?

Part 3: What is printed br, dr_1, and dr_2? What is the difference between the other two parts?