

1) Identify whether each of the following is either

- a. (a) memory leak
- b. (b) dangling pointer
- c. (c) unresolved pointer
- d. (d) something else

1a)

```
10 int main(int argc, const char * argv[]) {
11     int *ptr = (int *)malloc(sizeof(int) * 10);
12
13     *ptr = 100;
14     *(ptr + 9) = 10;
15
16     return 0;
17 }
```

1b)

```
10 int main(int argc, const char * argv[]) {
11     int *ptr;
12
13     int x = 10;
14
15     int y = 20;
16
17     int sum = x + y;
18
19     *ptr = 2140;
20
21     return 0;
22 }
```

1c)

```
10 int main(int argc, const char * argv[]) {
11     char *ptr = (char *)malloc(sizeof(char));
12     char *par = par;
13
14     free(ptr);
15
16     *ptr = 'f';
17
18     ptr = (char *)malloc(sizeof(char));
19
20     free(par);
21
22     return 0;
23 }
```

2) What is wrong with this code snippet?

```
9
10 struct Node {
11     int data;
12     struct Node *next;
13 };
14
15 void addNode(struct Node **head, int data) {
16     struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
17     newNode->data = data;
18     newNode->next = *head;
19     *head = newNode;
20 }
21
22 void traverseList(struct Node *head) {
23     struct Node *current = head;
24     while (current != NULL) {
25         printf("%d\n", current->data);
26         current = current->next;
27     }
28 }
29
30 void inOrderTraversal(struct Node *head) {
31     struct Node *current = head;
32     while (current != NULL) {
33         printf("%d\n", current->data);
34         current->next = NULL;
35         current = current->next;
36     }
37 }
38
39 int main(int argc, const char * argv[]) {
40     struct Node *head = NULL;
41     addNode(&head, 1);
42     addNode(&head, 2);
43     addNode(&head, 3);
44
45     inOrderTraversal(head);
46
47     return 0;
48 }
```

3) What operation do we use to add to a stack?

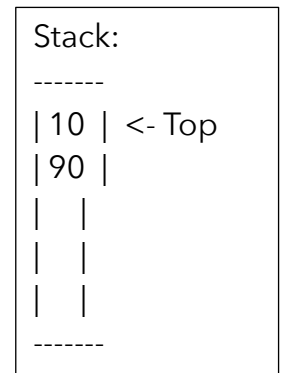
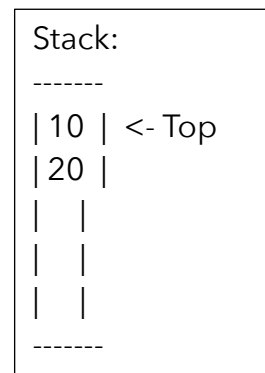
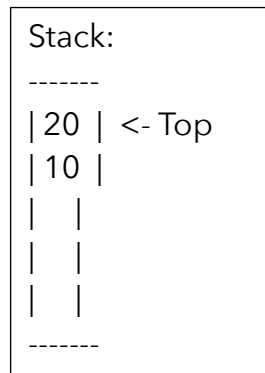
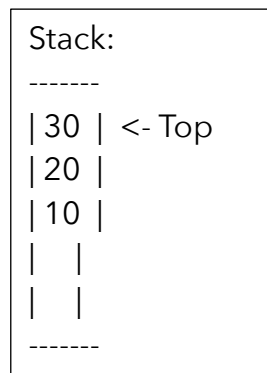
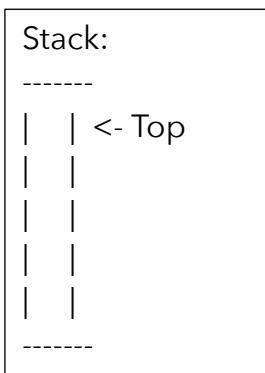
4) How do we access the 10th element in a 50-element stack?

5) Which of the following allows us to copy a list of nodes?

```
15 void copyList(Node ** oldL, Node ** newL) {  
16  
17 }  
18  
19 void copyList(Node * oldL, Node * newL) {  
20  
21 }  
22  
23 void copyList(Node *& oldL, Node *& newL) {  
24  
25 }
```

6) Are shallow or deep copies better?

7) Below each stack, declare what operations were performed



- 8) [Easy] There is nothing wrong with the semantics of the code (it is valid C++ code). Pretend that pop and push are implemented correctly. For simplicity, we use a counter. There is something wrong with the code because it does not return true for a valid set of code. A valid code is considered on with a balanced amount of open (and closing) in addition to a balanced count of { and }. You may need to add, remove, or modify functions to complete this.

```
12  static int count = 0;
13
14  void pop() {
15      --count; // Assume we pop here
16  }
17
18  void push() {
19      ++count; // Assume we push here
20  }
21
22  bool isValidCode(string code) {
23      for (auto &c : code) {
24          if (c == '(' || c == '{') {
25              push();
26          } else {
27              pop();
28          }
29      }
30
31      return count == 0;
32  }
```

9) Write push, pop, and peak function for some stack

10) What is an ADT?

11) Why do we have a call stack? Is it possible to have custom unwinding?