Quiz 4 Version C

Due No due date **Points** 20 **Questions** 12
Available after Nov 14 at 3:30pm **Time Limit** None
Allowed Attempts Unlimited

Instructions

Quiz for Lecture 4: Linked Data Structures, Linked Lists, and const

18 points required to pass

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	25 minutes	4 out of 20

Score for this attempt: **4** out of 20
Submitted Nov 14 at 4:01pm
This attempt took 25 minutes.

Question 1 — 0 / 1 pts

Examine this code. Line numbers in main are specified in comments. In the check boxes below, check each line of code that WILL NOT COMPILE.

```
struct cl_node {   char data;   cl_node *next;   };

int main() {

    cl_node n1{ 'a', nullptr };          // 2

    cl_node *p1 = &n1;                   // 3
```

```
    const cl_node *p2(&n1);          // 4
    p1->data = 'x';                  // 5
    p2->data = 'n';                  // 6
    const cl_node n2(n1);            // 7
    cout << n2.data << '\n';         // 8
    p1 = &n2;                        // 9
}
```

- [ ] line 2 will NOT compile
- [ ] line 3 will NOT compile

'ou Answered

- [x] line 4 will NOT compile

- [ ] line 5 will NOT compile

Correct!

- [x] line 6 will NOT compile

p2 is a pointer-to cl_node const, so it is not legal to change a member of the cl_node object indirectly through p2, even though the cl_node object that p2 currently points to (n1) is not const

'ou Answered

- [x] line 7 will NOT compile

- [ ] line 8 will NOT compile

Correct!

- [x] line 9 will NOT compile

it is not legal for a pointer-to (non-const) cl_node to pointer to a cl_node const object

Question 2

0 / 1 pts

Suppose we have this structure definition, for a node in a singly linked list of **char**:

struct cl_node { char data; cl_node *next; };

Which of these would be the most reasonable interface for a function that makes a singly linked list of **char** from a C-string?

'ou Answered

- (•) **cl_node *mk_clist_from_Cstring(cl_node *p) { ... }**
- ( ) void mk_clist_from_Cstring(const char *cstr) { ... }
- ( ) char *mk_clist_from_Cstring(const char *cstr) { ... }

orrect Answer

- ( ) **cl_node *mk_clist_from_Cstring(const char *cstr) { ... }**
- ( ) char *mk_clist_from_Cstring(const char *cstr) { ... }
- ( ) void mk_clist_from_Cstring(const char *cstr) { ... }
- ( ) void mk_clist_from_Cstring(cl_node *p) { ... }

Question 3

0 / 1 pts

Suppose we have this structure definition, for a node in a singly linked list of **char**:

struct cl_node { char data; cl_node *next; };

Which of these would be the most reasonable interface for a function

that modifies the data in one or more of the linked list nodes, but does not modify the shape of the linked list?

- void clist_mod_data(cl_node **pp) { ... }

orrect Answer

- **void clist_mod_data(cl_node *p) { ... }**

'ou Answered

- **void clist_mod_data(const cl_node *p) { ... }**

Question 4

1 / 1 pts

Suppose we have this structure definition, for a node in a singly linked list of **int**:

struct il_node { int data; il_node *next; };

Which of these would be the most reasonable interface for a function that accesses the data in one or more of the linked list nodes, but does not modify any data and does not modify the shape of the list?

Correct!

- **int ilist_access_data(const il_node *p) { ... }**
- int ilist_access_data(il_node **pp) { ... }
- int ilist_access_data(il_node *p) { ... }

Question 5

1 / 1 pts

Suppose we have this structure definition, for a node in a singly linked list

of **double**:

struct dl_node { double data; dl_node *next; };

Which of these would be the most reasonable interface for a function that modifies the shape of the linked list (adding one or more nodes, deleting one or more nodes, changing the order of nodes, ...)?

- void dlist_change_shape(const dl_node *p) { ... }
- **void dlist_change_shape(dl_node **pp) { ... }** — Correct!
- void dlist_change_shape(dl_node *p) { ... }

Question 6

2 / 2 pts

Suppose we have this structure definition, for a node in a singly linked list of **int**:

struct il_node { int data; il_node *next; };

Write the body of code for this function, that returns the number of nodes in a list of integers:

size_t ilist_size(const il_node *p)

{

// this is the part you have to write as your answer

}

Your Answer:

size = 0;

// Pretty much want to iterate through the whole loop and update size each
// iteration

```
while ( p != nullptr ) {
  size += 1;
  p = p->next;
}
return size;
```

```
size_t count(0);
for ( ; p; p = p->next)
    count += 1;
return count;
```

Question 7

0 / 2 pts

Suppose we have this structure definition, for a node in a singly linked list of **char**:

struct cl_node { char data; cl_node *next; };

Write the body of code for this function, that makes a list of **char** from a C-string (that is, an array of **char** terminated with a null **char**) and returns the address of the first node in the list:

```
cl_node *mk_clist_from_Cstring(const char *s)
{
    // this is the part you have to write as your answer
}
```

Your Answer:

```
// a recursive implementation
if (*s == '\0')
    return nullptr;
return new cl_node{ *s, mk_clist_from_Cstring(s + 1) };
```

see implementation in Lecture 4 notes, for example

Question 8

0 / 2 pts

Suppose we have this structure definition, for a node in a singly linked list of **char**:

struct cl_node { char data; cl_node *next; };

Write the body of code for this function, that deletes the first node in the list. If the list is empty, the function should do nothing.

```
void clist_delete_head(cl_node **pp)
{
    // this is the part you have to write as your answer
```

```
}
```

Your Answer:

```
// iterative solution
if (*pp == nullptr)
    return;
cl_node *tmp(*pp);
*pp = (*pp)->next;
delete tmp;
```

Question 9

0 / 2 pts

Suppose we have this structure definition, for a node in a singly linked list of **int**:

struct il_node { int data; il_node *next; };

Write the body of code for this function, that returns the sum of all the data values in the list (or **0**, if the list is empty):

```
int ilist_sum(const il_node *p)
{
    // this is the part you have to write as your answer
}
```

Your Answer:

```
// an iterative implementation
int sum(0);
for ( ; p; p = p->next)
    sum += p->data;
return sum;

// a recursive implementation
if (!p)
    return 0;
return p->data + ilist_sum(p->next);
```

Question 10

0 / 2 pts

Suppose we have this structure definition, for a node in a singly linked list of **char**:

struct cl_node { char data; cl_node *next; };

Write the body of code for this function, that deletes the list of characters and sets the header node in the calling function to **nullptr** (or **0**) when done.

```
void clist_delete(cl_node **pp)
{
    // this is the part you have to write as your answer
```

```
}
```

Your Answer:

```
// iterative solution
while (*pp) {
    cl_node *tmp(*pp);
    *pp = (*pp)->next;
    delete tmp;
}
```

Question 11

0 / 2 pts

Suppose we have this structure definition, for a node in a singly linked list of **double**:

struct dl_node { double data; dl_node *next; };

Write the body of code for this function, that divides the value of each **double** in the list by **x**, where **x** is a value passed in by the calling function. If **x** is **0.0**, this function should do nothing.

```
void dlist_divide_by_x(dl_node *p, double x)
{
    // this is the part you have to write as your answer
}
```

Your Answer:

```
if (x == 0.0)
    return;
for ( ; p; p = p->next)
    p->data /= x;
```

Question 12

0 / 3 pts

Suppose we have this structure definition, for a node in a singly linked list of **int**:

struct il_node { int data; il_node *next; };

Write the body of code for this function, that moves the first node in the list of **int** to the end of the list, that is, rotates the elements by one place. A list that started as 1, 2, 3, 4 should be 2, 3, 4, 1 when this function returns. If the list is empty or only has one node, this function should do nothing.

```
int ilist_rotate_one(il_node **pp)
{
    // this is the part you have to write as your answer
}
```

Your Answer:

```
if (*pp == nullptr || (*pp)->next == nullptr)
    return;   // empty or one-node list: do nothing
// keep track of the first node
cl_node *first = *pp;
first->next = nullptr;
// modify the list to start with the second node
*pp = (*pp)->next
// find the end
for ( ; (*pp)->next; pp = &(*pp)->next)
    ;
// attach original first node to end
(*pp)->next = first;
```

Quiz Score: **4** out of 20