

Announcements

Tenure-track Search Status Report

Campus Visits

Course Scheduling

"Thursday" Extras

Need for confidentiality!!!

Box-and-pointer Diagrams

List-traversal Examples

Program Management

Header files (example: `MyroC`, `stdio.h`)

Implementation files

Compiling

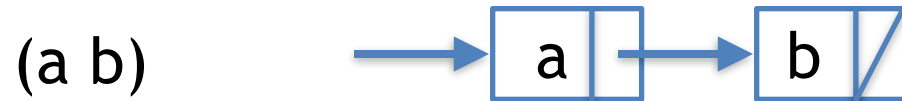
Office Hours (as usual this week—as least for now)

Monday, Wednesday: 1:00-3:00 pm

Tuesday, Wednesday, Friday: 2:30-4:00, as usual

Box-and-pointer Diagrams

Review: diagrams for several lists



What is the proper diagram for the null list: () ?

A. NULL

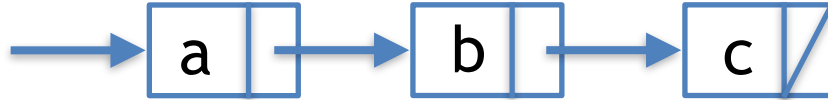


C. something else

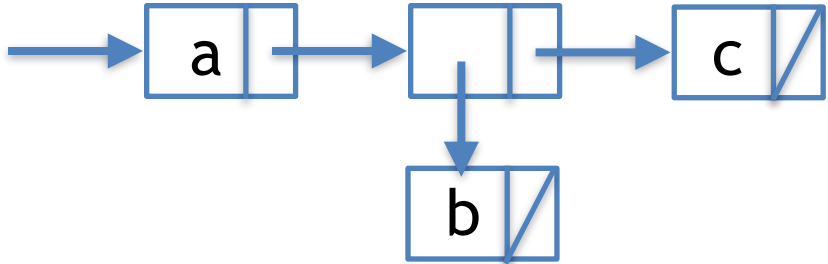
Box-and-pointer Diagrams

What is an appropriate diagram for (a (b) c)?

A.



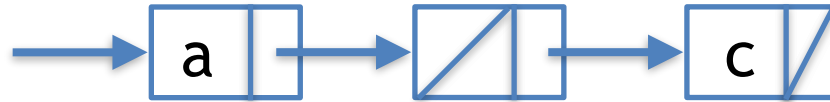
B.



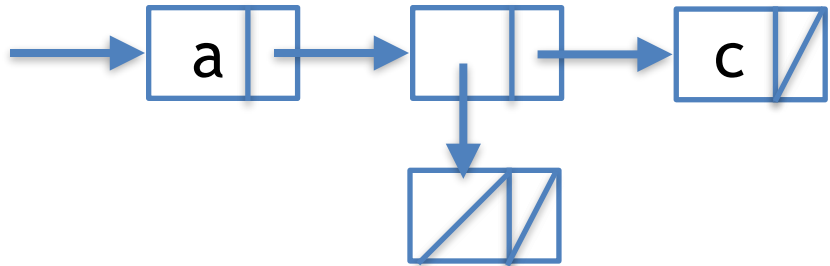
Box-and-pointer Diagrams

What is an appropriate diagram for (a () c)?

A.



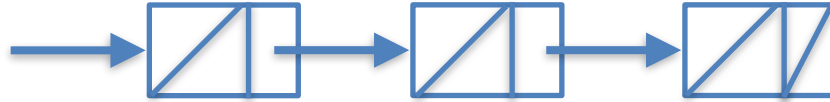
B.



C. something else

Box-and-pointer Diagrams

How many elements are on the following list?



- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

How would you represent this list, using Scheme's notation with parentheses?

List traversal procedures for finding the sum of node values:
The following two are proposed:

```
int sum (struct node * ptr) {  
    int total = 0;  
    while (ptr != NULL) {  
        total += ptr->data;  
        ptr = ptr-> next;  
    }  
    return total;  
}
```

```
int sum (struct node * ptr) {  
    struct node * temp = ptr;  
    int total = 0;  
    while (temp != NULL) {  
        total += temp->data;  
        temp = temp-> next;  
    }  
    return total;  
}
```

Suppose these are used on two lines in a program:

```
int sum1 = sum (first);
```

```
int sum2 = sum (first);
```

When will `sum1 == sum2` (for all lists)

- | | | | |
|---------------------------|-----------------------|-----------------------|------------|
| A. Both left
and right | B. Left, not
right | C. Right, not
left | D. Neither |
|---------------------------|-----------------------|-----------------------|------------|

List traversal procedures for finding the sum of node values:
Two more approaches are proposed:

```
int sum (struct node ** ptr) {  
    int total = 0;  
    while (*ptr != NULL) {  
        total += (*ptr)->data;  
        *ptr = (*ptr)-> next;  
    }  
    return total;  
}
```

```
int sum (struct node ** ptr) {  
    struct node * temp = *ptr;  
    int total = 0;  
    while (temp != NULL) {  
        total += temp->data;  
        temp = temp-> next;  
    }  
    return total;  
}
```

Suppose these are used on two lines in a program:

```
int sum1 = sum (&first);
```

```
int sum2 = sum (&first);
```

When will `sum1 == sum2` (for all lists)

- | | | | |
|---------------------------|-----------------------|-----------------------|------------|
| A. Both left
and right | B. Left, not
right | C. Right, not
left | D. Neither |
|---------------------------|-----------------------|-----------------------|------------|

C allows a program to be divided into several pieces:

- function prototypes (header files —.h)
- one or more prototype implementation files
- a main program

Several reasons might be suggested for dividing a program:

1. Multiple files allow programs to be organized logically
2. Different files can be compiled at different times (as long as the final results are linked before a program is run)
3. Parts of a program can be reused easily
4. When one part of a program is changed, only that part can be recompiled before linking and running a program

How many of these reasons can make a substantial difference in managing large programs?

- | | | | | |
|--------------------------------|-----------------------------|-----------------------------|-----------------------------|-------------------------------|
| A. none of
these
reasons | B. 1 of
these
reasons | C. 2 of
these
reasons | D. 3 of
these
reasons | E. all of
these
reasons |
|--------------------------------|-----------------------------|-----------------------------|-----------------------------|-------------------------------|

The make utility works with a file Makefile to aid program management. Some possible reasons to use this capability:

1. "make" is easier to type than "gcc ... "
2. Makefile can record all needed libraries, avoiding typos
3. Makefile can handle both compiling of separate files and linking all compiled files together
4. Makefile can keep track of what needs to be recompiled when some file changes
5. Makefile can manage deletion of old files, without risking deletion of essential files one does not want deleted
6. Makefile can execute sequences of Linux commands

A. 2 of
these
reasons

B. 3 of
these
reasons

C. 4 of
these
reasons

D. 5 of
these
reasons

E. all of
these
reasons