Lecture 12

Monday, May 15, 2017 12:30 PM

ITERATORS CONTD, STACKS

Problem Solving with Computers-I

https://ucsb-cs24-sp17.github.io/

Monday, May 15, 2017 12:30 PM

2

How is pa04 going?

A. Done
B. I am on track to finish
C. I am passing test1()
D. Having trouble with test1()
E. Haven't started

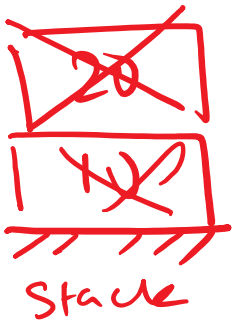
Monday, May 15, 2017 12:30 PM

3

Stacks – container class available in the C++ STL

- Container class that uses the Last In First Out (LIFO) principle
- Methods
  i. push()
  ii. pop()
  iii. top()
  iv. empty()

return true if stack is empty



push(10)
push(20)
top() return 20
pop()
pop()

Demo reversing a string, and review of lab06 code

Monday, May 15, 2017 12:30 PM

4

Notations for evaluating expression

- Infix number operator number (7 + (3 * 5)) – (4 / 2)
- Prefix operators precede the operands
- Postfix operators come after the operands

Infix

7+3

7+(3*5)

Prefix

+73

+*35-7

Postfix

73+

735*+

Monday, May 15, 2017 12:30 PM

5

Lab06 – part 1: Evaluate a fully parenthesized infix expression

(4 * ((5 + 3.2) / 1.5)) // okay

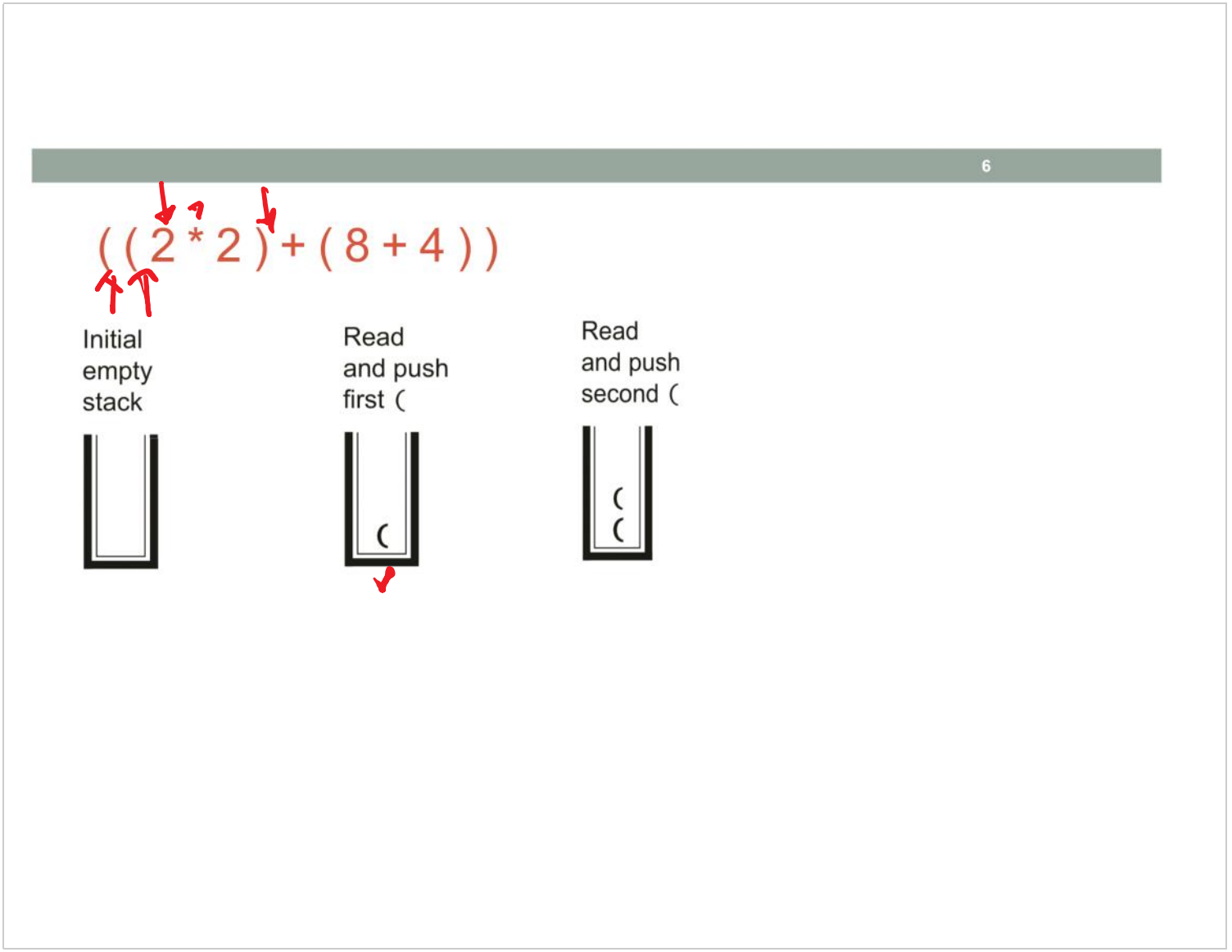
(4 * ((5 + 3.2) / 1.5) // unbalanced parens - missing last ')'

(4 * (5 + 3.2) / 1.5)) // unbalanced parens - missing one '('

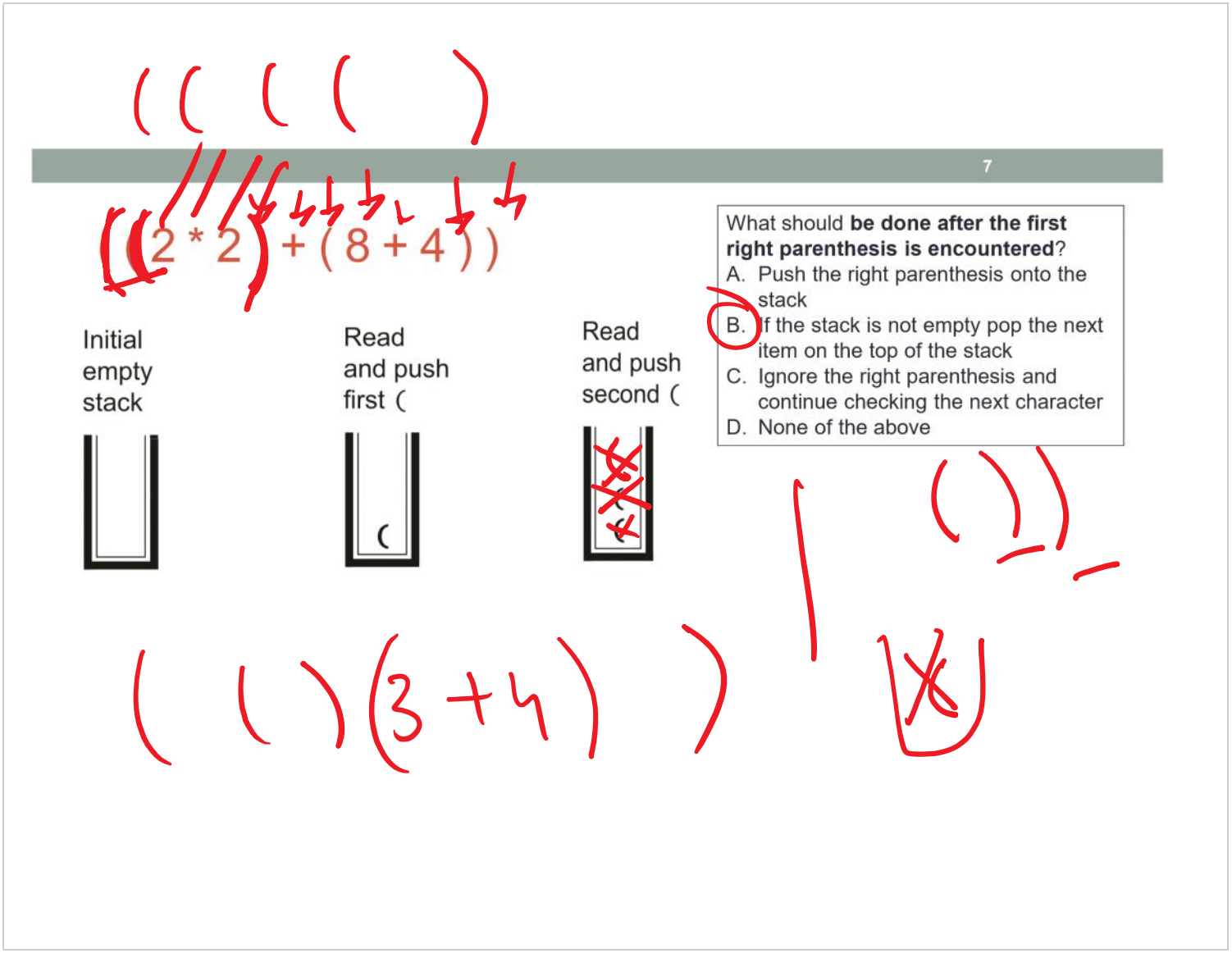
4 * ((5 + 3.2) / 1.5) // not fully-parenthesized at '*' operation

(4 * (5 + 3.2) / 1.5) // not fully-parenthesized at '/' operation

Monday, May 15, 2017 12:30 PM

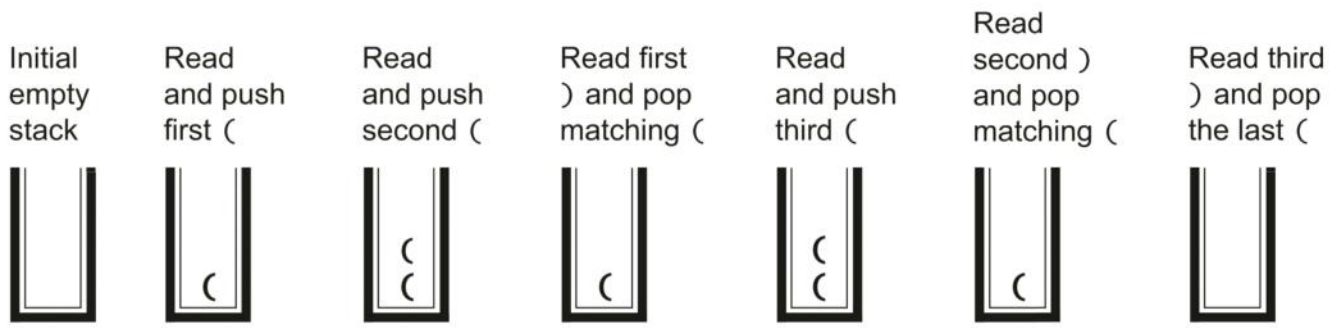


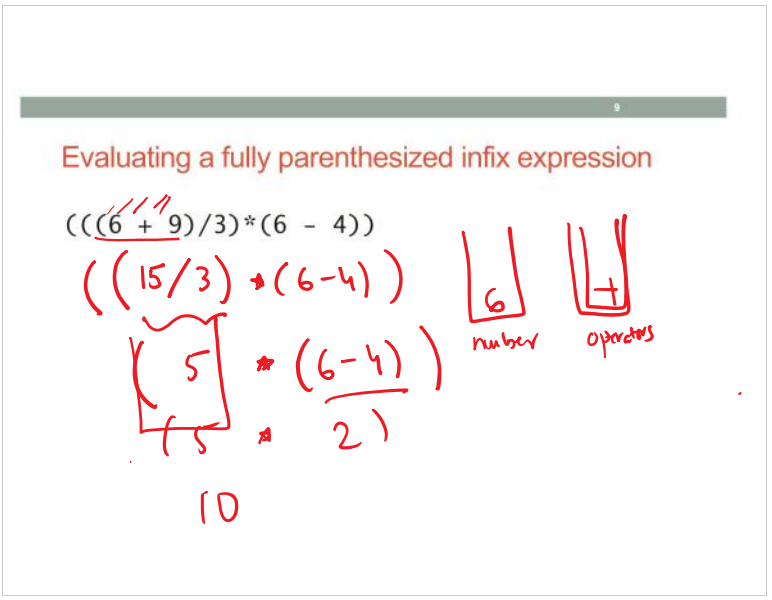
Monday, May 15, 2017 12:30 PM

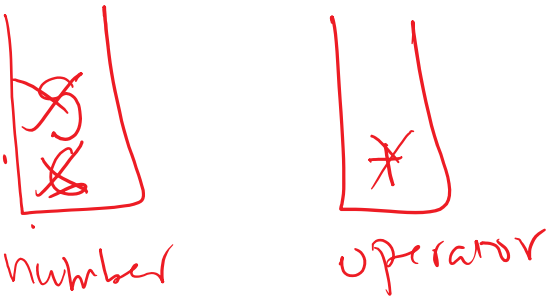


Monday, May 15, 2017 12:30 PM

((2*2)+(8+4))




9
Evaluating a fully parenthesized infix expression
(((6 + 9)/3)*(6 - 4))
((15/3)*(6-4))
(5 * (6-4))
(5 * 2)
10
6
+
number
operators


number
operator


((6 + (9 / 3)) * (6 - 4))
6 + 9


2
9
num
oprv

9 / 3

6 + 3

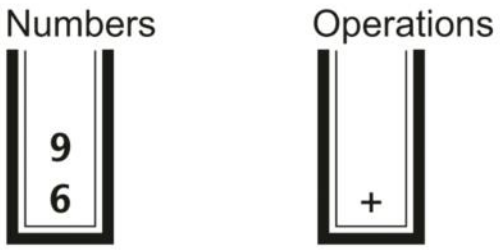
((12 + 6) / 7)

Monday, May 15, 2017 12:30 PM

Evaluating a fully parenthesized infix expression

Characters read so far (shaded):

(((6 + 9) / 3) * (6 - 4))



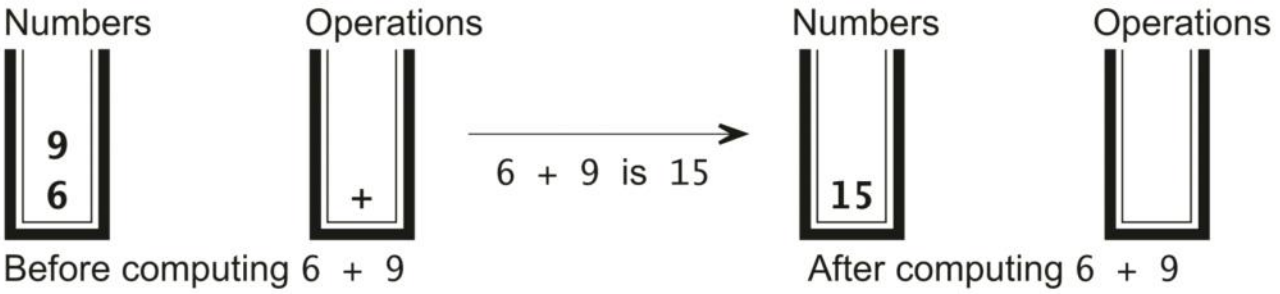
Monday, May 15, 2017 12:30 PM

11

Evaluating a fully parenthesized infix expression

Characters read so far (shaded):

(((6 + 9) / 3) * (6 - 4))



Before computing 6 + 9

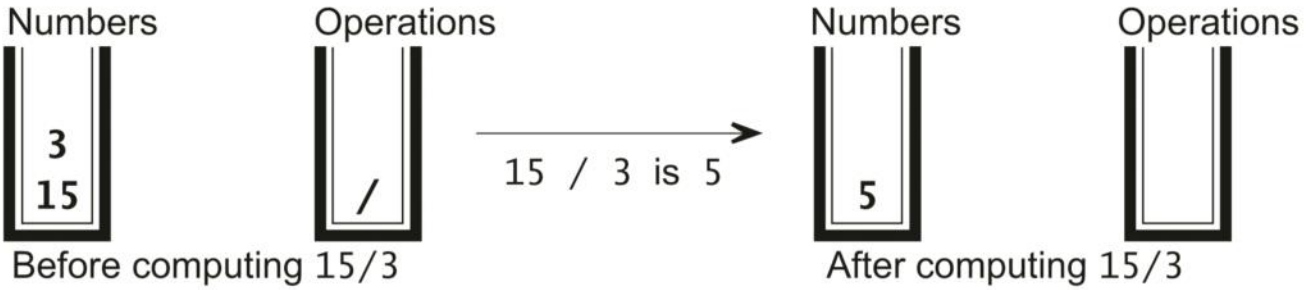
After computing 6 + 9

Monday, May 15, 2017 12:30 PM

Evaluating a fully parenthesized infix expression

Characters read so far (shaded):

(((6 + 9) / 3) * (6 - 4))



Before computing 15/3 After computing 15/3

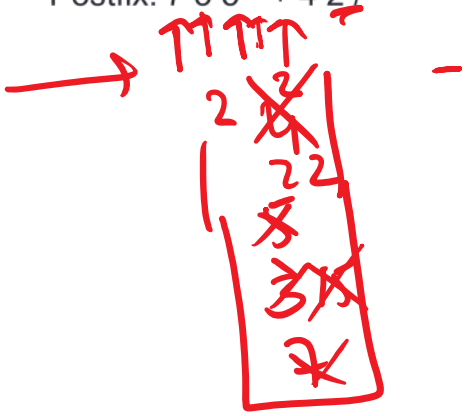
Monday, May 15, 2017 12:30 PM

13

Evaluating post fix expressions using a single stack

Postfix: 7 3 5 * + 4 2 / -

Infix: (7 + (3 * 5)) – (4 / 2)



22 - 2

3 → 5

7 + 15

Pop out of the stack every time you encounter an operator

Monday, May 15, 2017 12:30 PM

C++ Iterators

- Iterators are generalized pointers.
- Let's consider a very simple algorithm (printing in order) applied to a very simple data structure (sorted array)

10	20	25	30	46	50	55	60

```
void print_inorder(int* p, int size) {
    for(int i=0; i<size; i++) {
        std::cout << *p << std::endl;
        ++p;
    }
}
```

- We would like our print "algorithm" to also work with other data structures
- How should we modify it to print the elements of a LinkedList?

Monday, May 15, 2017 12:30 PM

15

C++ Iterators

Consider our implementation of LinkedList

```
void print_inorder(LinkedList<int> *p, int size)  {
    for(int i=0; i<size; i++)
    {
        std::cout << *p <<std::endl;
        ++p;
    }
}
```

When will the above code work?

A. The operator "<<" is overloaded to print the data key of a LinkedList Node
B. The LinkedList class overloads the ++ operator
C. Both A and B
D. None of the above

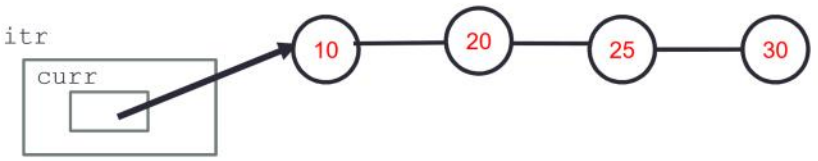
16

C++ Iterators

- To solve this problem the LinkedList class has to supply to the client (print_inorder) with a generic pointer (an iterator object) which can be used by the client to access data in the container sequentially, without exposing the underlying details of the class

```
void print_inorder(LinkedList<int>& ll)  {
  LinkedList<int>::iterator itr = ll.begin();
  LinkedList<int>::iterator en = ll.end();

  while(itr!=en)
    {
        std::cout << *itr <<std::endl;
        ++itr;
    }
}
```



Monday, May 15, 2017 12:30 PM

Demo

- Provide an iterator to the linkedList template class written in last lecture

Monday, May 15, 2017 12:30 PM

18

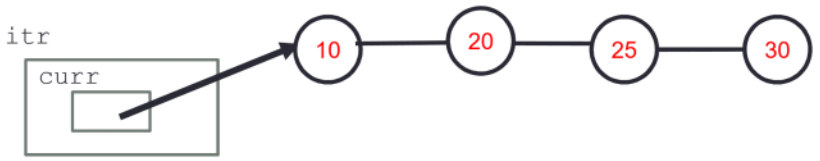
C++ Iterators

```
void print_inorder(LinkedList<int>& ll)  {
  LinkedList<int>::iterator itr = ll.begin();
  LinkedList<int>::iterator en = ll.end();

  while(itr!=en)
    {
        std::cout << *itr <<std::endl;
        ++itr;
    }
}
```

What should **begin()** return?

A. The address of the first node in the linked list container class
B. An iterator type object that contains the address of the first node
C. None of the above



Monday, May 15, 2017 12:30 PM

19

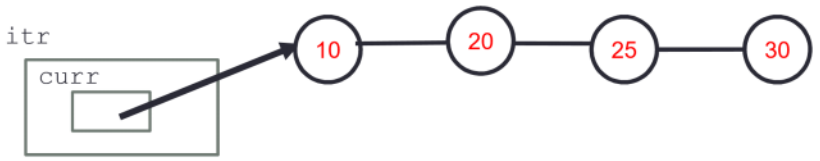
C++ Iterators

```
void print_inorder(LinkedList<int>& ll)  {
  LinkedList<int>::iterator itr = ll.begin();
  LinkedList<int>::iterator en = ll.end();

  while(itr!=en)
    {
        std::cout << *itr <<std::endl;
        ++itr;
    }
}
```

List the operators that the iterator has to implement?

A. *
B. ++
C. !=
D. All of the above
E. None of the above



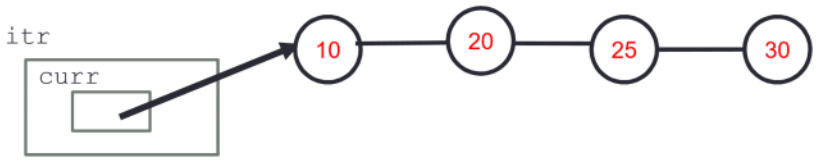
20

C++ Iterators

```
void print_inorder(LinkedList<int>& ll)  {
  LinkedList<int>::iterator itr = ll.begin();
  LinkedList<int>::iterator en = ll.end();

  while(itr!=en)
    {
        std::cout << *itr <<std::endl;
        ++itr;
    }
}
```

How should the diagram change as a result of the statement ++itr; ?



21

C++ Iterators

```
void print_inorder(LinkedList<int>& ll)  {
  auto itr = ll.begin();
  auto en = ll.end();

  while(itr!=en)
    {
        std::cout << *itr <<std::endl;
        ++itr;
    }
}
```

How should the diagram change as a result of the statement ++itr; ?