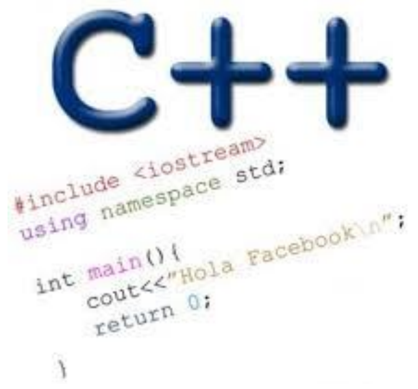
ITERATORS CONTD, STACKS

Problem Solving with Computers-I

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



```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```

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

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()

Demo reversing a string, and review of lab06 code

Notations for evaluating expression

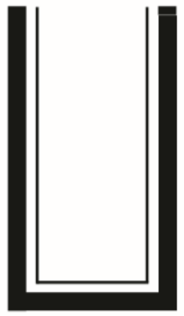
- Infix number operator number (7 + (3 * 5)) – (4 / 2)
- Prefix operators precede the operands
- Postfix operators come after the operands

Lab06 – part 1: Evaluate a fully parenthesized infix expression

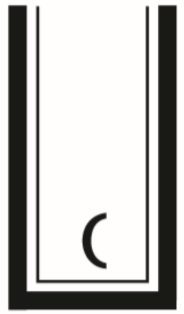
```
( 4 * ( ( 5 + 3.2 ) / 1.5 ) ) // okay

( 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
```

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

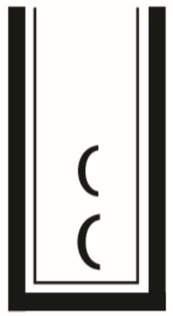
Initial empty stack



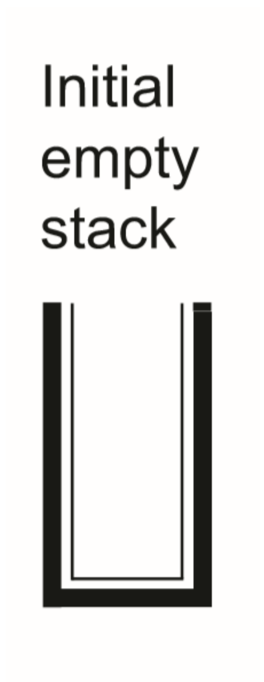
Read and push first (

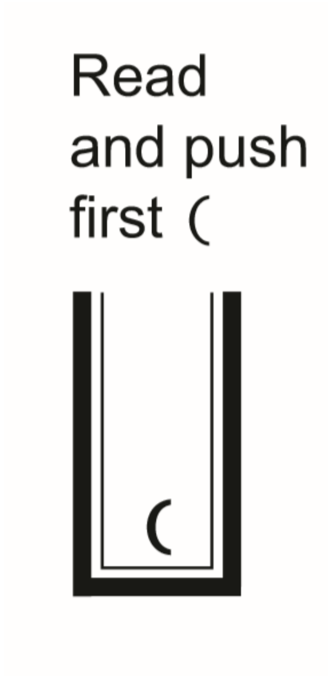


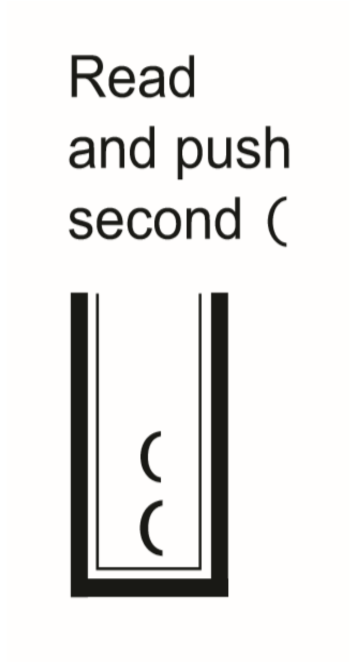
Read and push second (



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



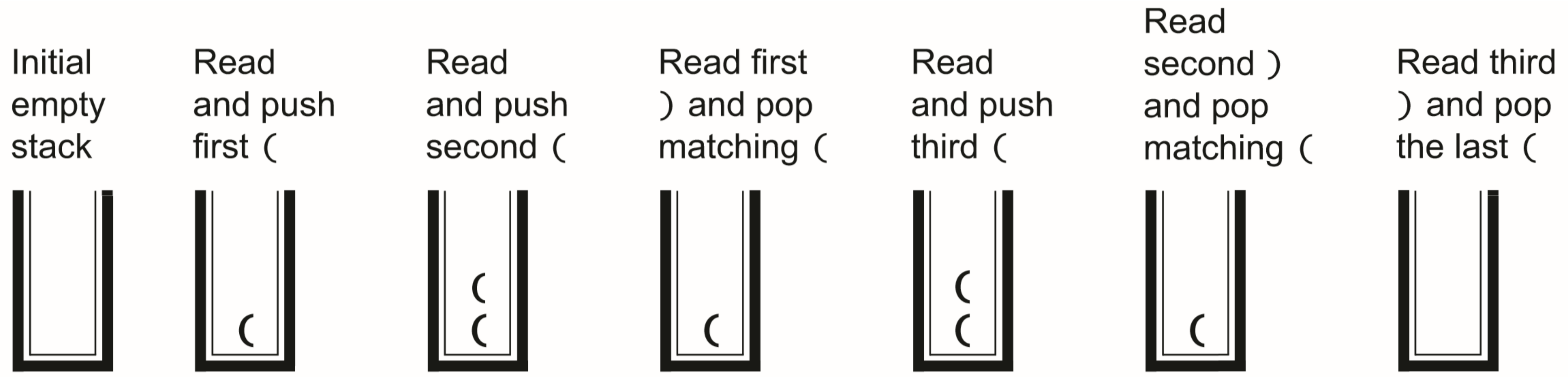




What should **be done after the first right parenthesis is encountered**?

A. Push the right parenthesis onto the stack
B. If the stack is not empty pop the next item on the top of the stack
C. Ignore the right parenthesis and continue checking the next character
D. None of the above

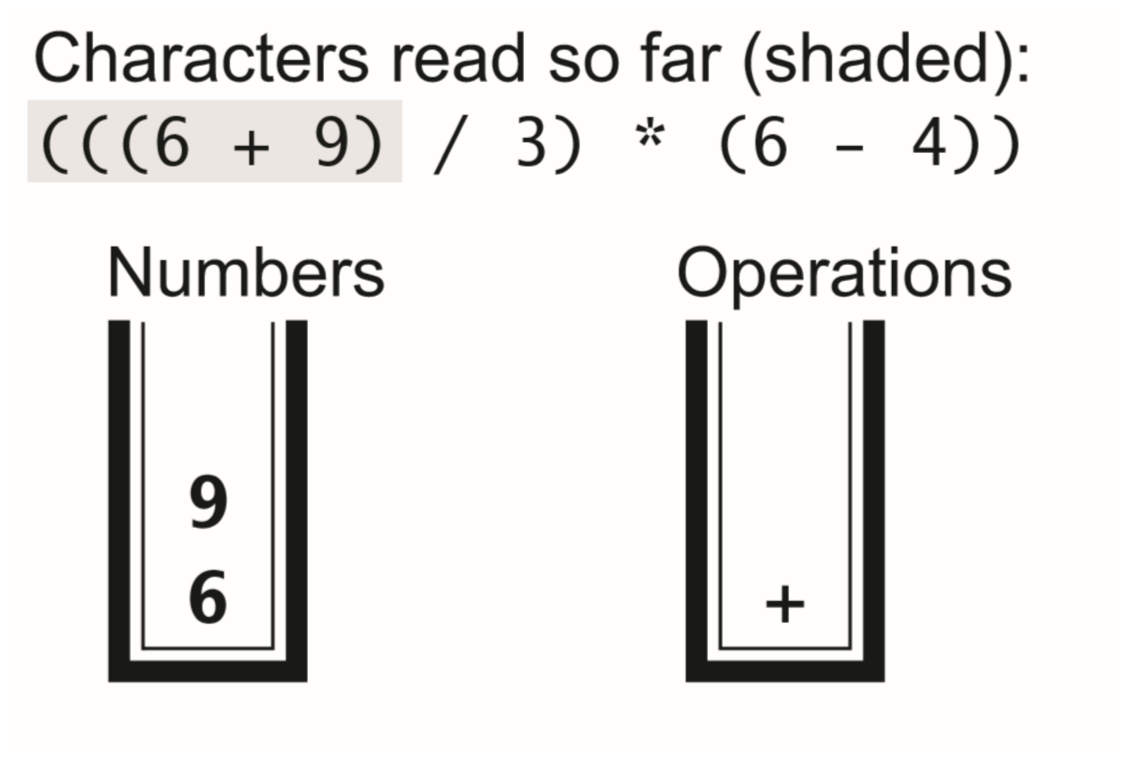
((2*2)+(8+4))


Initial empty stack
Read and push first (
Read and push second (
Read first) and pop matching (
Read and push third (
Read second) and pop matching (
Read third) and pop the last (

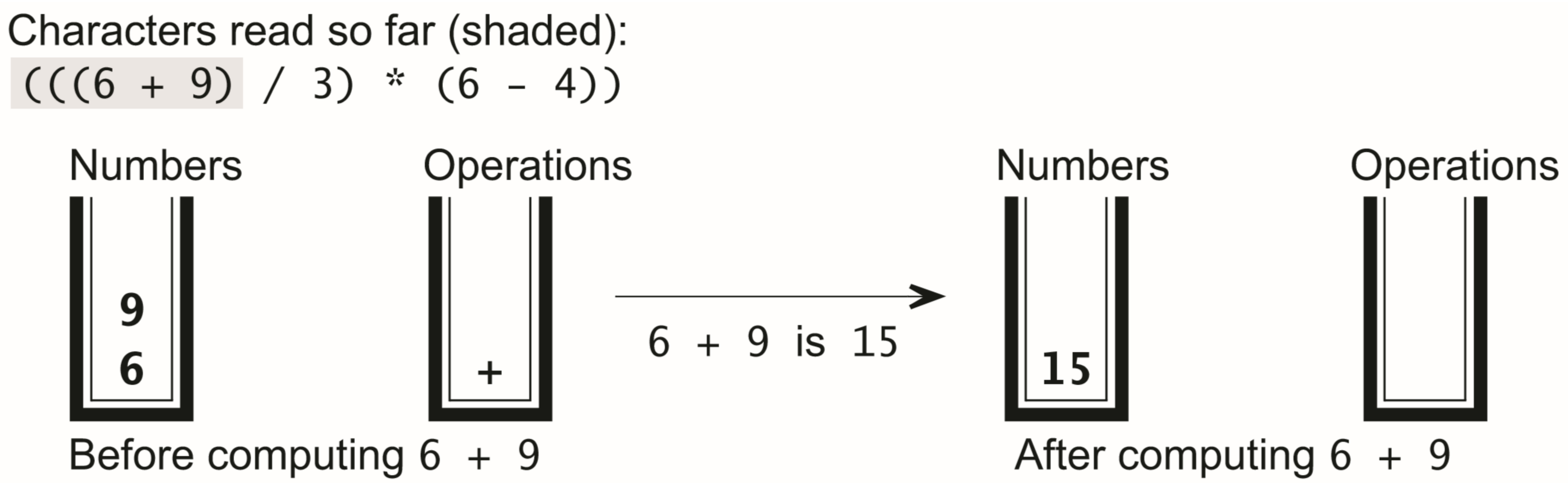
Evaluating a fully parenthesized infix expression

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

Evaluating a fully parenthesized infix expression



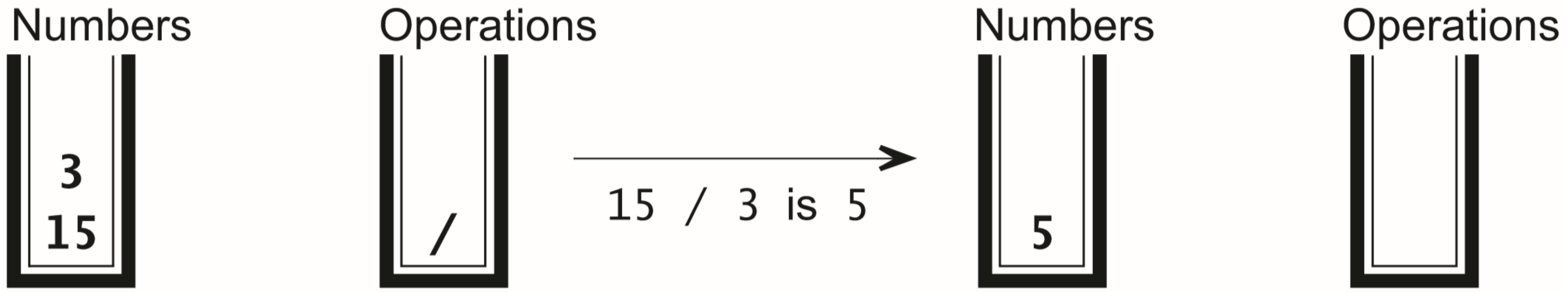
Evaluating a fully parenthesized infix expression



Evaluating a fully parenthesized infix expression

Characters read so far (shaded):

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



Before computing 15/3

After computing 15/3

Evaluating post fix expressions using a single stack

Postfix: 7 3 5 * + 4 2 / -

Infix: (7 + (3 * 5)) – (4 / 2)

C++ Iterators

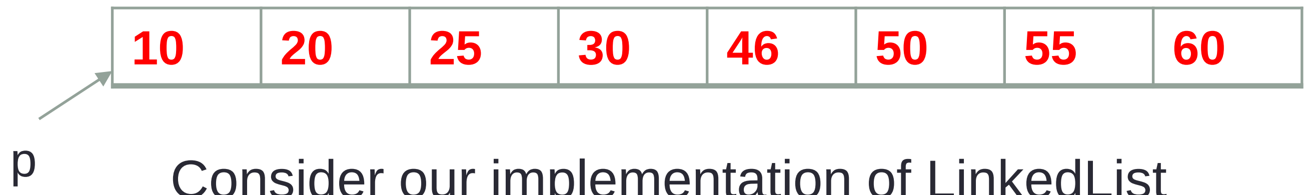
- Iterators are generalized pointers.
- Let's consider a very simple algorithm (printing in order) applied to a very simple data structure (sorted array)

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?

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;
    }
}
```

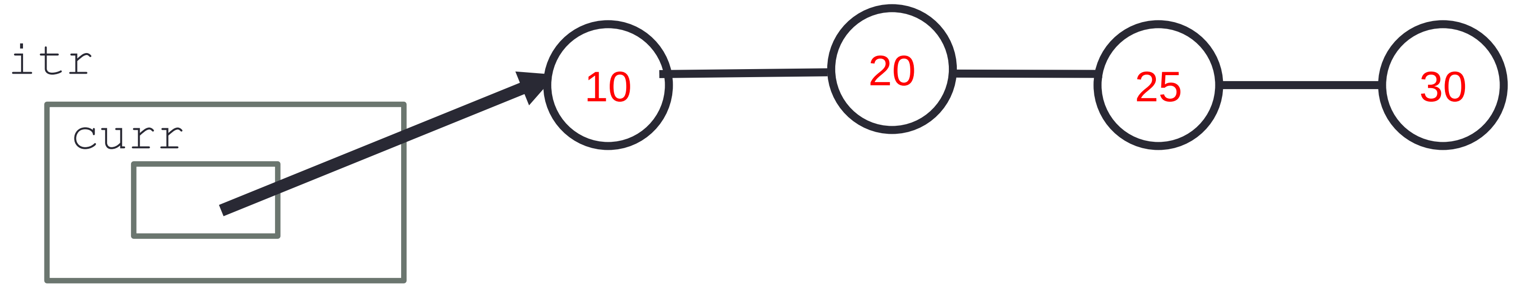
When will the above code work?

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

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;
    }
}
```



Demo

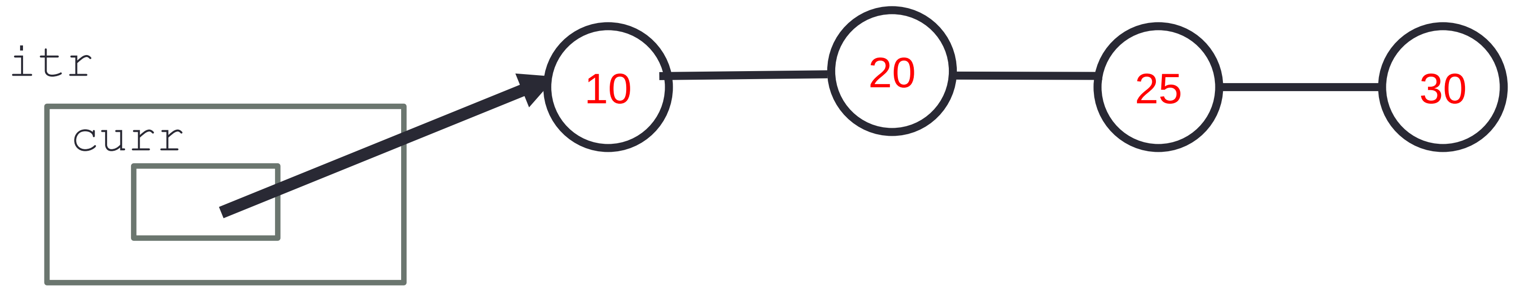
- Provide an iterator to the linkedList template class written in last lecture

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

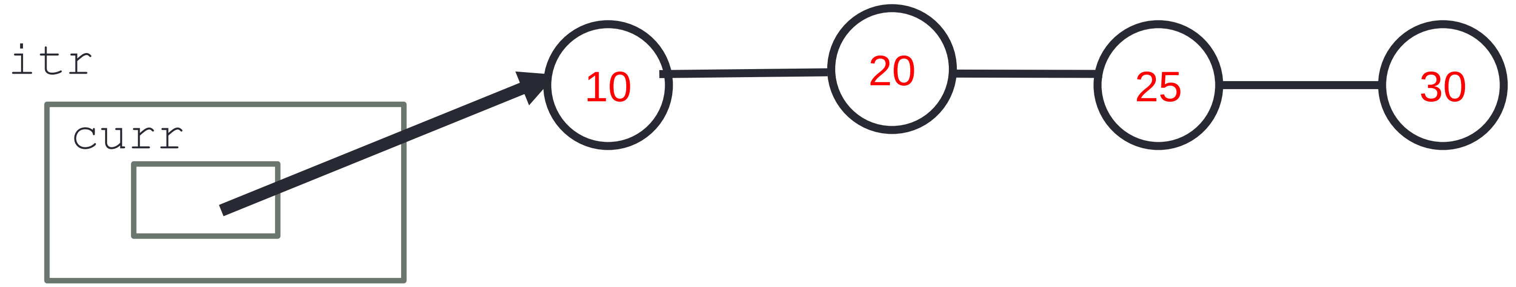


C++ Iterators

```
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?

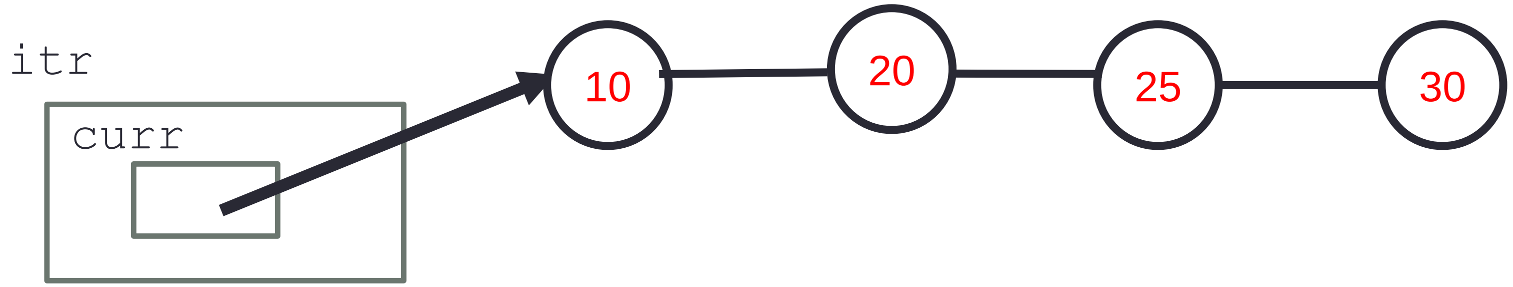
A. *
B. ++
C. !=
D. All of the above
E. None of the above



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;
    }
}
```

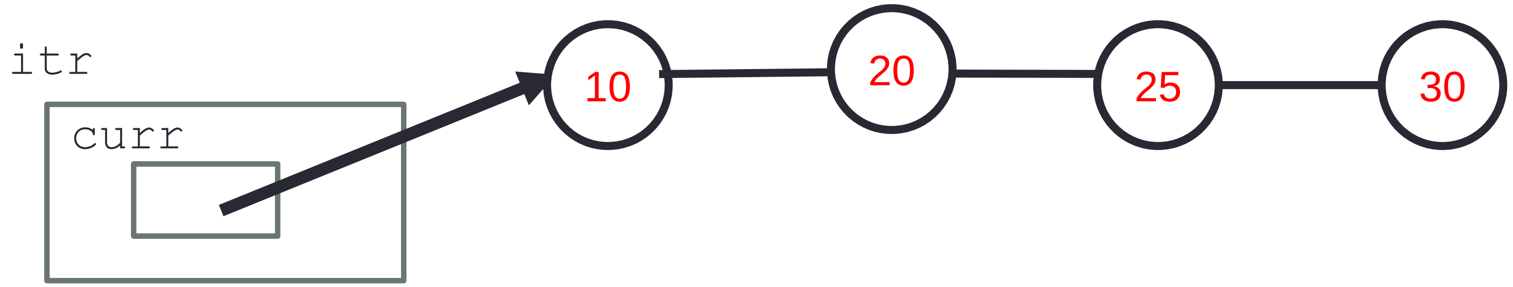
How should the diagram change as a result of the statement ++itr; ?



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; ?


itr
curr
10
20
25
30