TEMPLATES AND ITERATORS

Problem Solving with Computers-I

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

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

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

Announcements

- Checkpoint deadline for pa04 (aka lab05) is due today at 11:59pm
  - Be sure to push your code to github AND
  - Submit on submit.cs

How is pa04 going?

A. Going well
B. I am working on passing test1()
C. I am stuck and don’t know how to proceed
D. I haven’t started

Demo

- Converting the intList class (linked list) to a class that uses templates

Linked list, with templates:

```
template<class Item>

class Node {
public:
  Node<Item>* next;
  Item const data;

  Node( const Item & d ) :
     data(d) {
     next = 0;
  }

};
```

How would you create **a Node object** on the runtime stack?

int myInt =10;

A. Node n(myInt);
B. Node<int> n;
C. Node<int> n(myInt);
D. Node<int> n = new Node<int>(myInt);

Linked list, with templates:

```
template<class Item>

class Node {
public:
  Node<Item>* next;
  Item const data;

  Node( const Item & d ) :
     data(d) {
     next = 0;
  }

};
```

Write a line of code to create a new Node object with int data on the heap and make nodePtr to point to it.

```
int myInt=10;
```

Automatic type deduction with “auto”

Linked list, with templates:

```
template<class Item>

class Node {
public:
  Node<Item>* next;
  Item const data;

  Node( const Item & d ) :
     data(d) {
     next = 0;
  }

};
```

```
auto p = new Node<int>(myInt);
```

CHANGING GEARS: C++STL

- The C++ Standard Template Library is a very handy set of three built-in components:
  - Containers: Data structures
  - Iterators: Standard way to search containers
  - Algorithms: These are what we ultimately use to solve problems

Motivation for iterators

- The same algorithms can be applied to multiple container classes
- C++ STL avoids rewriting the same algorithm for different container classes by using ITERATORS
- Algorithms interface with containers via iterators

STL container classes

```
                         array
                        vector
                         deque
                  forward_list
                          list
                         stack
                         queue
                priority_queue
                           set
multiset (non unique keys)
                 unordered_set
                           map
                 unordered_map
                      multimap
                        bitset
```

C++ Iterators

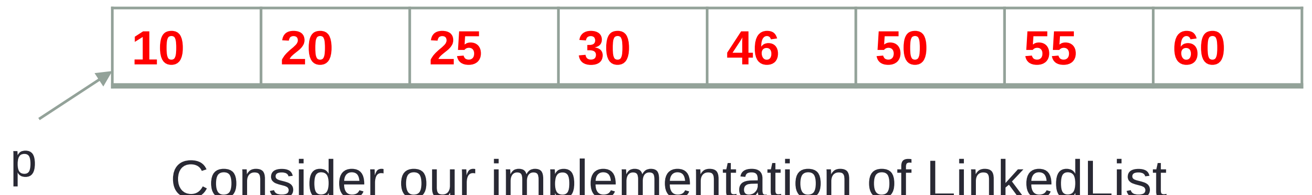
- Iterators are generalized pointers.
- Let’s consider a very simple algorithm (printing in order) applied to a very simple data structure (sorted array)

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?

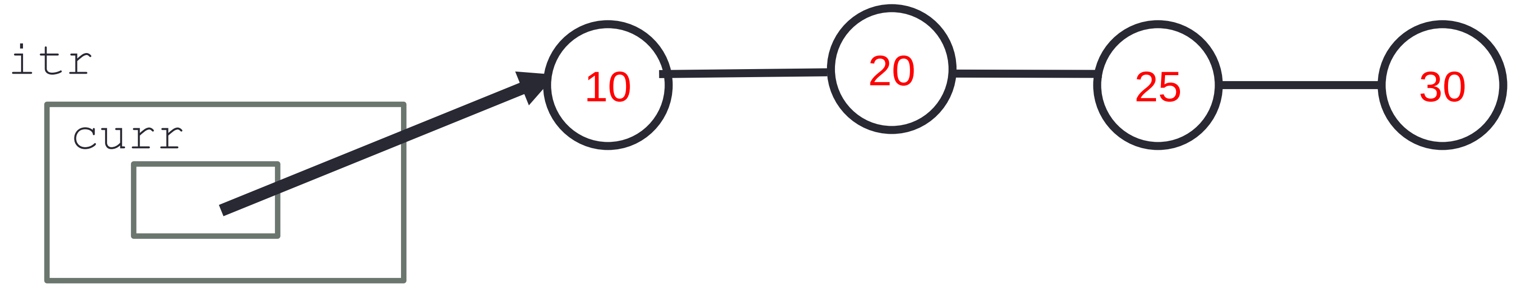
A. The operator “<<“ is overloaded to print the data key of a LinkedList Node
B. The LinkedList class overloads the ++ operator
C. Both A and B
D. None of the above

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 it = ll.begin();
  LinkedList<int>::iterator en = ll.end();

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



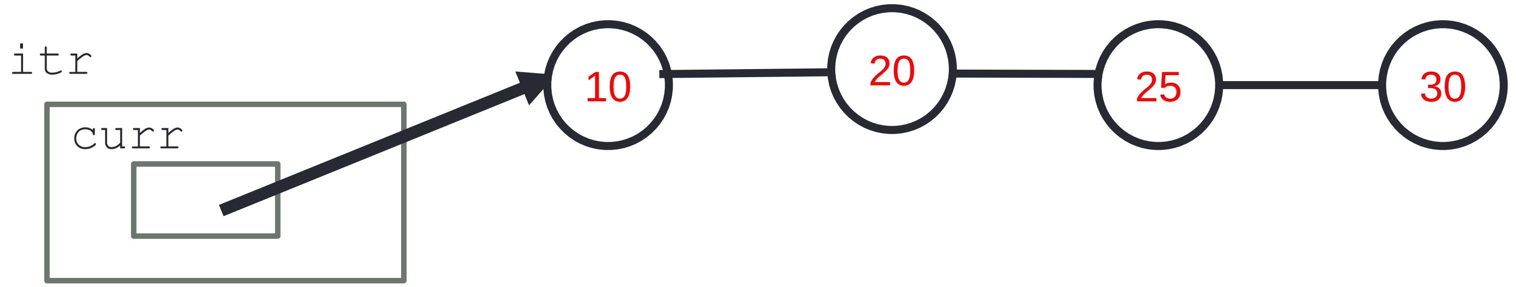
C++ Iterators

```
void print_inorder(LinkedList<int>& ll)   {
  LinkedList<int>::iterator it = 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



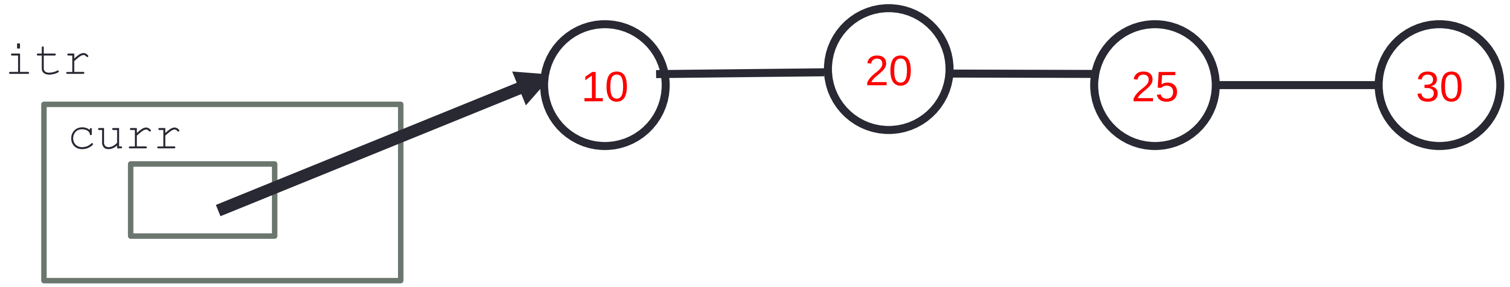
C++ Iterators

```
void print_inorder(LinkedList<int>& ll)  {
  LinkedList<int>::iterator it = 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

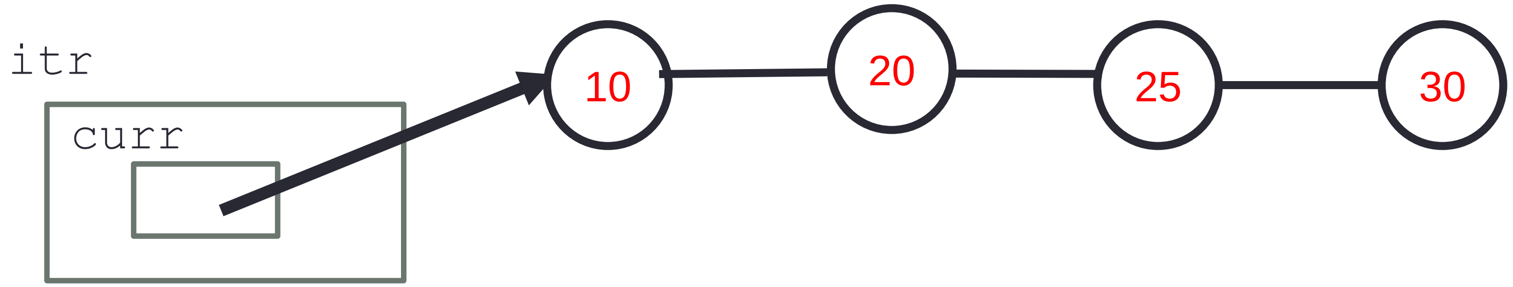


C++ Iterators

```
void print_inorder(LinkedList<int>& ll)  {
  LinkedList<int>::iterator it = 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; ?

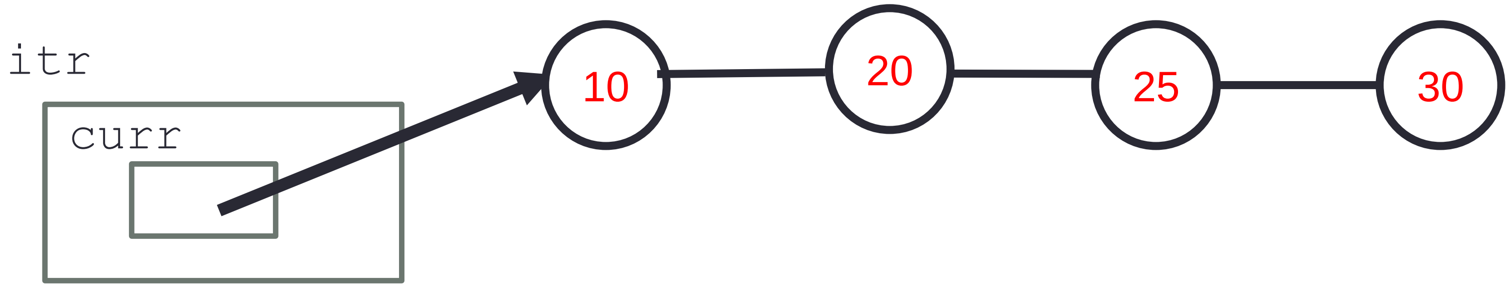


C++ Iterators

```
void print_inorder(LinkedList<int>& ll)  {
  auto it = 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