COMPARISON CLASSES AND GENERIC POINTERS

Problem Solving with Computers-II

From last class….

```
int main(){
      int arr[]={10, 2, 80};
      priority_queue<int> pq;
      for(int i=0; i < 3; i++)
            pq.push(arr[i]);

      while(!pq.empty()){
            cout<<pq.top()<<endl;
            pq.pop();
      }
      return 0;
}
```

What is the output of this code?

A. 10 2 80
B. 2 10 80
C. 80 10 2
D. 80 2 10
E. None of the above

Comparison class

- A class used to perform comparisons.
- Implements a function operator that compares two keys

```
class cmp{
        bool operator()(int& a, int& b) const {
                return a > b;
         }
};

//Use cmp to compare any two keys
cmp foo;
cout<<foo(x, y);
```

priority queue using the compare class to order the keys

```
class Comp{
        bool operator()(int& a, int& b) const {
                return a > b;
        }
};
```

Comp is a *Compare* class, defined such that it returns `true` if its first argument comes *before* its second argument in a weak ordering. But because the priority queue outputs largest elements first, the elements that "come before" are actually output last. That is, the front of the queue contains the "last" element according to the weak ordering imposed by *Compare*.

Configure PQ with a compare class

```
class Comp{
        bool operator()(int& a, int& b) const {
              return a > b;
        }
};
int main(){
      int arr[]={10, 2, 80};
      priority_queue<int, vector<int>, Comp> pq;
      for(int i=0; i < 3; i++)
            pq.push(arr[i]);

      while(!pq.empty()){
            cout<<pq.top()<<endl;
            pq.pop();
      }
      return 0;
}
```

What is the output of this code?

A. 10 2 80

B. 2 10 80

C. 80 10 2

D. 80 2 10

E. None of the above

std::priority_queue template arguments

The template for priority_queue takes 3 arguments:

```
template <
      class T,
      class Container= vector<T>,
      class Compare = less <T>
      > class priority_queue;
```

- The first is the type of the elements contained in the queue.
- If it is the only template argument used, the remaining 2 get their default values:
  - a **vector<T>** is used as the internal store for the queue,
  - **less is a comparison** class that provides priority comparisons. See earlier slide for definition of the Compare parameter

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

C++ Iterators

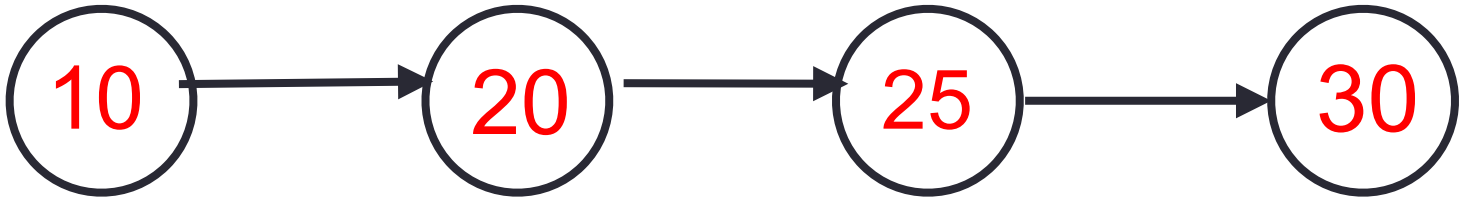
- Iterators act like pointers but for data structures.
- Let's consider how we generally use pointers to parse an array

10	20	25	30	46	50	55	60

```
void printElements(int arr[], int size) {
    int* p= arr;
    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
- E,g Linked list or BST

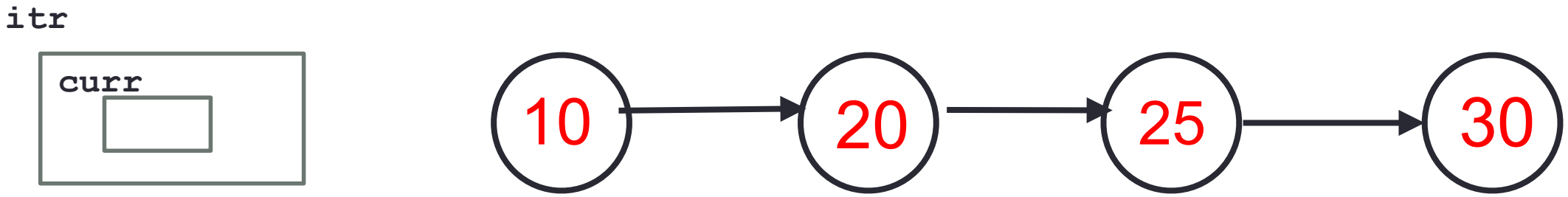
Can a similar pattern work with a LinkedList? Why or Why not?




```
void printElements(LinkedList& ll, int size) {
    ____________________________________//How should we define p?
    for(int i=0; i<size; i++) {
        std::cout << *p << std::endl;
        ++p;
    }
}
```

C++ Iterators

- To solve this problem the LinkedList class has to supply to the client (printElements) 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



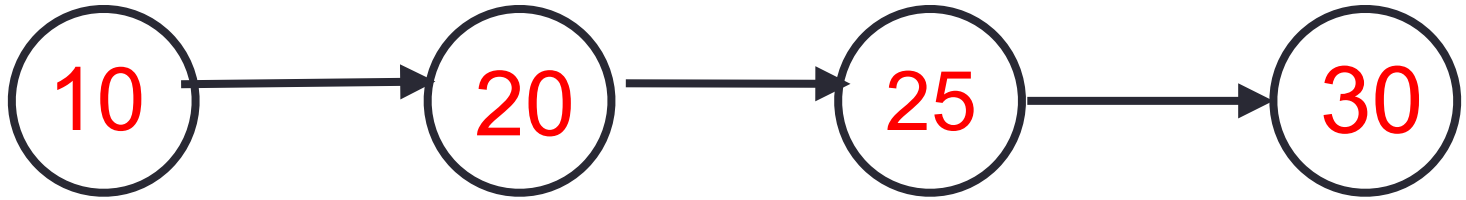

```
void printElements(LinkedList& ll)  {
  LinkedList::iterator itr = ll.begin();
  LinkedList::iterator en = ll.end();
  while(itr!=en){
        std::cout << *itr <<" ";
        ++itr;
  }
  cout<<endl;
}
```

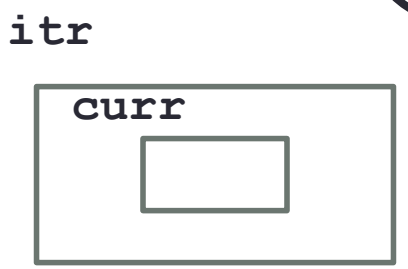
C++ Iterators: Initializing the iterator

```
void printElements(LinkedList& ll)  {
  LinkedList::iterator itr = ll.begin();
  LinkedList::iterator en = ll.end();
  while(itr!=en){
        std::cout << *itr <<" ";
        ++itr;
  }
  cout<<endl;
}
```

What is the return value of **begin()** ?

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



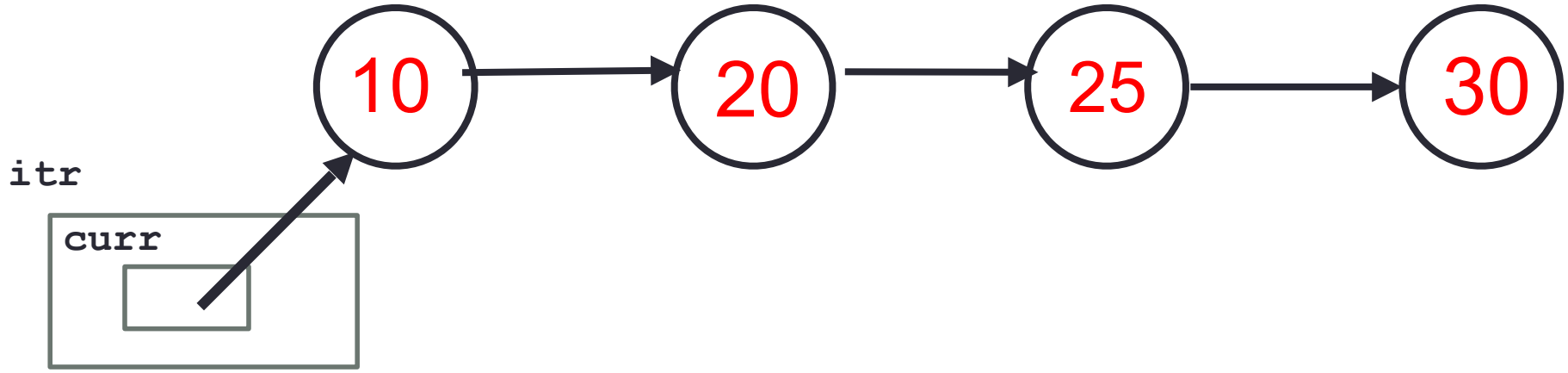


C++ Iterators: Overloading operators

```
void printElements(LinkedList& ll)  {
  LinkedList::iterator itr = ll.begin();
  LinkedList::iterator en = ll.end();
  while(itr!=en){
        std::cout << *itr <<" ";
        ++itr;
  }
  cout<<endl;
}
```

List the operators that must be overloaded for iterator objects?

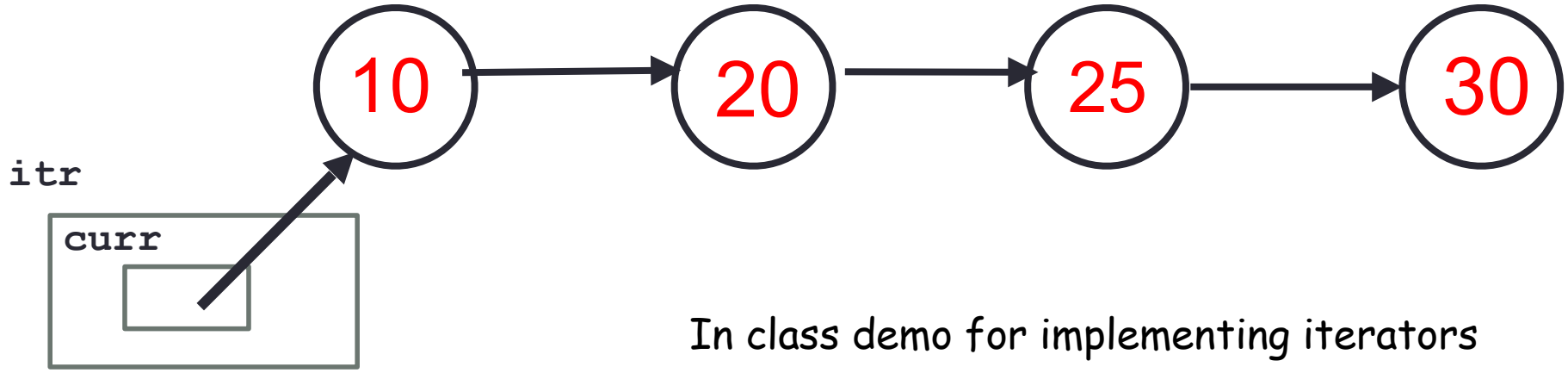
A. *
B. ++
C. !=
D. All of the above
E. None of the above



C++ Iterators

```
void printElements(LinkedList& ll)  {
  LinkedList::iterator itr = ll.begin();
  LinkedList::iterator en = ll.end();
  while(itr!=en){
        std::cout << *itr <<" ";
        ++itr;
  }
  cout<<endl;
}
```

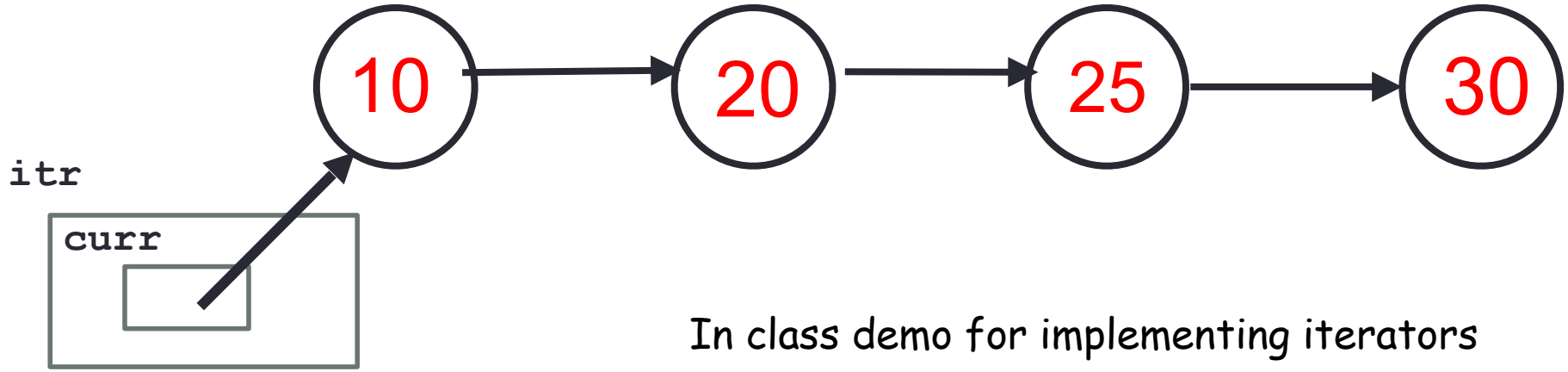
How should the diagram change as a result of the statement ++itr; ?



In class demo for implementing iterators

C++ shorthand: auto

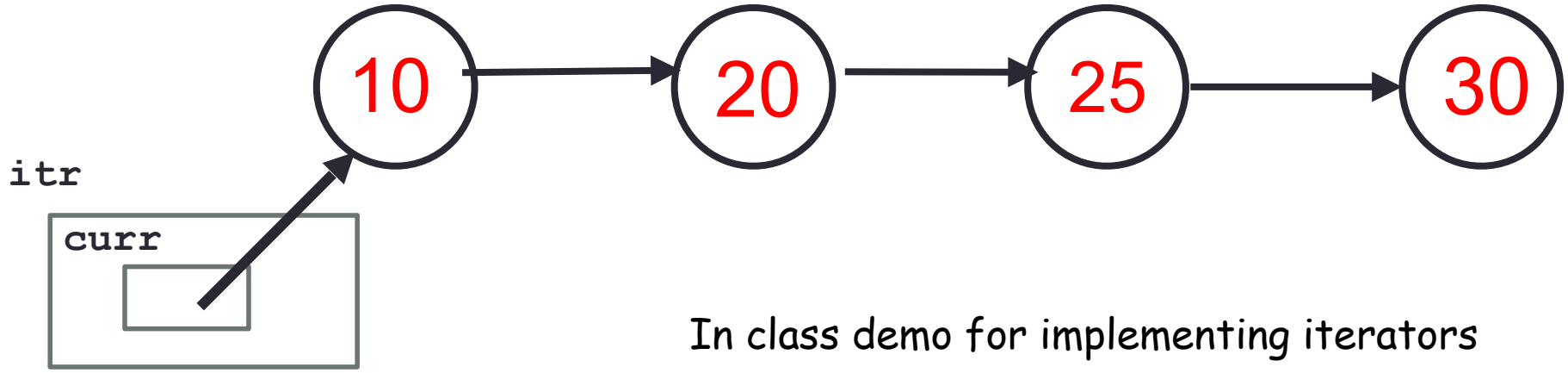
```
void printElements(LinkedList& ll)  {
  auto itr = ll.begin();
  auto en = ll.end();
  while(itr!=en){
        std::cout << *itr <<" ";
        ++itr;
  }
  cout<<endl;
}
```



In class demo for implementing iterators

Finally: unveiling the range based for-loop

```
void printElements(LinkedList& ll)  {
  for(auto item:ll){
        std::cout << item <<" ";
  }
  cout<<endl;
}
```



In class demo for implementing iterators

Challenge problems

```
int main(){
      int arr[]={10, 2, 80};
      priority_queue<int*> pq;
      for(int i=0; i < 3; i++)
            pq.push(arr+i);

      while(!pq.empty()){
            cout<<*pq.top()<<endl;
          pq.pop();
      }
      return 0;
}
```

What is the output of this code?

A. 10 2 80
B. 2 10 80
C. 80 10 2
D. 80 2 10
E. None of the above

Sort array elements using a pq storing pointers

```
int main(){
      int arr[]={10, 2, 80};
      priority_queue<int*> pq;
      for(int i=0; i < 3; i++)
            pq.push(arr+i);

      while(!pq.empty()){
            cout<<*pq.top()<<endl;
           pq.pop();
      }
      return 0;
}
```

How can we change the way pq prioritizes pointers?

Write a comparison class to print the integers in the array in sorted order

```
int main(){
      int arr[]={10, 2, 80};
      priority_queue<int*, vector<int*>, cmpPtr> pq;
      for(int i=0; i < 3; i++)
            pq.push(arr+i);

      while(!pq.empty()){
            cout<<*pq.top()<<endl;
           pq.pop();
      }
      return 0;
}
```