template <class List\_entry>

Error\_code List<List\_entry>::insert(int position, const List\_entry& x)

{

if (full())

return overflow;

if (position < 0 || position > count)

return scope\_error;

Node<List\_entry>\* new\_node, \* previous, \* following;

if (position > 0) {

previous = set\_position(position - 1);

following = previous->next;

}

else following = head;

new\_node = new Node<List\_entry>(x, following);

if (position == 0)

head = new\_node;

else

previous->next = new\_node;

count++;

return success;

}

template<class List\_entry>

Error\_code List<List\_entry>::retrieve(int position, List\_entry& x)const

{

if (empty())

return underflow;

if (position<0 || position>count)

return range\_error;

Node<List\_entry>\* previous;

if (position > 0) {

previous = set\_position(position - 1);

}

else previous = head;

new\_code = new Node<List\_entry>(x, previous);

x = new\_code.entry;

return x;

}

template<class List\_entry>

Error\_code List<List\_entry>::replace(int position, const List\_entry& x)

{

if (empty())

return underflow;

if (position<0 || position>count)

return range\_error;

Node<List\_entry>\* previous;

if (position > 0) {

previous = set\_position(position - 1);

}

else previous = head;

new\_code = new Node<List\_entry>(x, previous);

new\_code.entry=x;

return success;

}

template<class List\_entry>

Error\_code List<List\_entry>::remove(int position, List\_entry& x);

{

if (empty())

return underflow;

if (position<0 || position>count)

return range\_error;

Node<List\_entry>\* new\_node, \* previous, \* following;

new\_node = new Node<List\_entry>(x, following);

if (position > 0) {

previous = set\_position(position - 1);

following = new\_node->next;

}

else following = head;

count--;

return success;

}