Templates and Iterators

Linked List Data Structures

```
struct ListNode {
      Data data_;
      ListNode *next_;
      ListNode(Data d) : data_(d), next_(nullptr) {};
}

class LinkedList {
      ListNode *head_;
      …
public:
      …
```

What about a Linked List of something else?

```
struct ListNode {
      int data_;
      ListNode *next_;
      ListNode(int d) : data_(d), next_(nullptr) {};
}

class LinkedList {
      ListNode *head_;
      …
public:
      …
```

#include <vector>

Better Choice Templates

We have used templates such as in `std::vector` <int> or <Data> or ...

- Templates allow for generic code that is parameterized by types.

- Think of them as recipes to generate code

- With templates all code must be included as headers are included.
  - We use the following at the end of the header example for template version **`ll.h`**

```
// needed for template instantiation
#include "ll.cpp"
} // namespace snakelinkedlist
#endif //LL_H
```

Simple Template Function Example

```
template<typename T>
T smaller(T a, T b) {
    T smaller_value;
    smaller_value = ( a < b ) ? a : b;
    return smaller_value;
}
```

class

std::vector<int>

return ((a < b) ? a : b);

whatever

operator< says

cout << smaller<int>(1,2);

1

Now List

```
template<typename ElementType>
struct ListNode {
      ElementType data_;
      ListNode *next_;
      ListNode(ElementType d) : data_(d), next_(nullptr) {};
}

class LinkedList {
      ListNode<???> *head_;
      …
public:
      …
```

type of data container

Better Plan

```
template<typename ElementType>
class LinkedList {
  struct ListNode {
    ElementType data_;
    ListNode *next_;
    ListNode(ElementType d) : data_(d), next_(nullptr) {};
  }

  ListNode<ElementType> *head_;
  …
public:
  …
```

Template Insert at start from included .cpp

```
template<typename ElementType>
void LinkedList::InsertHead(ElementType new_data) {
     ListNode *new_node = new ListNode(new_data);
     new_node->next_ = head_;
     head_ = new_node;
}
```

How to access the linked list?

```
…
for(SnakeBody* curr = head_->next; curr; curr = curr->next) {
  ofRectangle body_rect(curr->position.x,
                        curr->position.y,
                        body_size_.x, body_size_.y);
  if (head_rect.intersects(body_rect)) {
    return true;
  }
}
```

node after head

check null

move to next node

vect = 1. GetVector() make a copy of 1 data as vector

vect[0] = vect[0]+1; modify the first node in vector

a) true

b) false

vect == 1. GetVector

modified vector

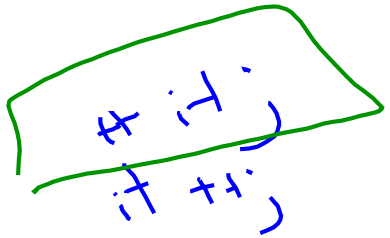
new copy of list as vector

STL style interface using range for?

```
…
for(auto snake_segment : snake_list_ )
  ofRectangle body_rect(snake_segment.position.x,
                        snake_segment.position.y,
                        body_size_.x, body_size_.y);
  if (head_rect.intersects(body_rect)) {
    return true;
  }
}
```

for (auto it = ll.begin(); it != ll.end(); ++it)

Iterators – Simple Forward Iterator Interface



Required Functions of iterator

- `Iterator()` – default constructor
- `Iterator & operator++()` – preincrement as next element
- `ElementType &operator*()` – dereference operator as to access
- `bool operator!=(const Iterator &rhs)` – not equal to detect end.

Required Functions of Linked List class

- `Iterator begin()` – return an iterator to the start of the list
- `Iterator end()` – return an iterator to past the end of the list

What to store in our iterator to access the list?

A. `ListNode *current_;`

B. `ListNode current_;`

C. `ListNode &current_;`

D. `LinkedList *current_;`

pointer to a ListNode

copy of a ListNode

reference to a ListNode

pointer to a List

Iterator & operator++()

x = 1
y = ++x;

```
Iterator & operator++() {
        if (current) {
            current = current->next;
        }

        return *this;
}
```

ElementType &operator*()

```
ElementType &operator*() {

}
```

return current->data_; or

return (*current).data_;

bool operator!=(const Iterator &rhs)

```
bool operator!=(const Iterator &rhs) {
    return current != rhs.current;


}
```

Iterator begin()

```
Iterator begin() {
        Iterator start;
            start.current = head_;
            return start;
}
```

Iterator end()

```
Iterator end() {
                Iterator stop;
                  stop.current = nullptr;
                return stop;
}
```