Computer Science 102
Lab 9

In this lab you will you will create C++ version of your linked list manager. You are *not required* to convert the list management components of your raytracer to C++ but you are free to do so if you wish. A sample list*main.cpp, list.h* and a skeleton *list.cpp* are provided for you.

Your mission is to complete the module named *list.cpp* which will contain the implementations of the following class methods.

```
class link_t
{
public:
   link_t(void);                  // default constructor
   link_t(void *);                // overloaded constructor
   void    set_next(link_t *);    // used in adding new link
   link_t *get_next(void);        // retrieve the next pointer
   void   *get_entity(void);      // retrieve entity pointer

private:
   link_t *next;                  // next link in the list
   void   *entity;                // entity managed by link
};

class list_t
{
public:
   list_t(void);                   // constructor
   void   add(void *);             // add entity to end of list
   void   start(void);             // set current to start of list
   void   *get_next(void);         // advance current pointer and
                                   // return entity it points to.

private:
   link_t *first;                 // first link
   link_t *last;                  // last link
   link_t *current;               // current link.
};
```

A discussion of the operation of the individual class methods follows and may be found in the class notes. A discussion of the operation of each method follows.

link_t methods

This constructor is passed a pointer to the entity that this new link will own. It should set the *next* pointer to NULL and set the *entity* pointer to the entity being passed in:

```
link_t::link_t(void *newentity)
{


}
```

The *set_next()* method is a typical “set” function that is used to tell the *link_t* to manipulate its own *next* pointer. It is called by the *add* method of the *list_t* class when an item that is not the first item is added to the list. It should set the *next* attribute of the *link_t* to *new_next;*

```
void link_t::set_next(link_t *new_next)
{


}
```

link_t *getter* methods

The *get_next()* method is a typical “get” function that is used as a way to tell the *link_t* to return the value of own *next* pointer.

```
link_t * link_t::get_next()
{


}
```

The *get_entity()* method is analogous. It would also work to simply make the *next* and *entity* elements *public*. Then any holder of a reference to the *link_t* could simply manipulate them directly... but it would be a *violation* of OO dogma to do so. It should return the *entity* pointer associated with the current link.

```
void * link_t::get_entity()
{


}
```

list_t class methods

The *list_t* class overrides the default constructor with its own constructor with no parameters:

```
list_t::list_t()
{
   first = NULL;
   last =  NULL;
   current = first;
}
```

Adding a new object to the list

The *add()* method creates a new *link_t* and passes its constructer a pointer to the entity to be attached to the *link_t.* The *link_t* constructor returns a pointer to the new *link_t.*

If this is the *first* item added to an empty list, the *list_add* method should set the *first, last* and *current* pointers to the new link.

Otherwise, the *next* pointer of the existing *last* link should be set to point to the new *link_t* and the *last* pointer of the *list_t* should be set to the new *link_t.*

```
void list_t::add(void *entity)
{
   link_t *link;



}
```

Retrieving the *first* element of the list

If the *list* is empty the *list_start()* method should return NULL. Otherwise, the *list_start* method sets the *current* pointer to the first *link* in the list and returns a pointer to the first *entity* in the list.

```
void * list_t::start(void)
{

}
```

Retrieving the next entity in the list.

The *get_next* method attempts to advance the *current* pointer. If the *current* pointer is already at the end of the list *NULL* will be returned. The use of the *persistent state variable current* will prove to be something of a pain in nested processing of the list. The function must return a pointer to the current *entity* in the list.

```
void_t * list_t::get_next(void)
{
   link_t *link;


}
```

In this lab you will submit a single file, list.cpp that includes the new class methods constructed as part of this lab.

```
sendlab.102.labsection#  lab#  list.cpp
```