CS 225

Data Structures

February 10 – Templates and Linked Memory

G Carl Evans

Abstract Class:

[Requirement]:

[Syntax]:

[As a result]:

virtual-dtor.cpp

```
 4 class Cube {
 5   public:
 6     ~Cube()// Print ~Cube() invoked.
       Cube()// Print Cube() invoked.
 7 };
 8
 9 class RubikCube : public Cube {
10   public:
11     ~RubikCube()// Print ~RubikCube()invoked.
       RubikCube()// Print RubikCube() invoked.
12 };
```

```
27 std::cout << "Non-virtual dtor:" << std::endl;
28 Cube *ptr = new RubikCube();
29 delete ptr;
```

virtual-dtor.cpp

```
15 class CubeV {
16   public:
17     virtual ~CubeV()// Print ~Cube() invoked.
18              CubeV()// Print Cube() invoked.
19 };
20
21 class RubikCubeV : public CubeV {
22   public:
23     ~RubikCubeV()// Print ~RubikCubeV()invoked.
24      RubikCubeV()// Print RubikCubeV() invoked.
25 };
```

```
31 std::cout << "Virtual dtor:" << std::endl;
32 CubeV *ptrV = new RubikCubeV();
33 delete ptrV;
```

```
[[gcevans@linux-a2 09-templates]$ ./virtual-dtor
Non-virtual dtor:
Cube() invoked.
RubikCube() invoked.
~Cube() invoked.

Virtual dtor:
CubeV() invoked.
RubikCubeV() invoked.
~RubikCubeV() invoked.
~CubeV() invoked.
[gcevans@linux-a2 09-templates]$
```

MP: Extra Credit

The most successful MP is an MP done early!

Unless otherwise specified in the MP, we will award +1 extra credit point per day **for completing Part 1** before the due date *(up to +7 points)*:

Example for mp stickers:

+7 points: Complete by **Monday**, Feb. 10 (11:59pm)
+6 points: Complete by **Tuesday**, Feb. 11 (11:59pm)
+5 points: Complete by **Wednesday**, Feb. 12 (11:59pm)
+4 points: Complete by **Thursday**, Feb. 13 (11:59pm)
+3 points: Complete by **Friday**, Feb. 14 (11:59pm)
+2 points: Complete by **Saturday**, Feb. 15 (11:59pm)
+1 points: Complete by **Sunday**, Feb. 16 (11:59pm)
mp stickers Due Date: Monday, Sept. 17

Abstract Data Type

List ADT

What types of “stuff” do we want in our list?

Templates

template1.cpp

```
1
2
3  T maximum(T a, T b) {
4    T result;
5    result = (a > b) ? a : b;
6    return result;
7  }
```

List.h

```
1  #pragma once
2
3
4  class List {
5    public:
6
7
8
9
10
11
12
13
14   private:
15
16
17
18 };
19
20
21
22
```

List.hpp

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

List Implementations

1.

2.

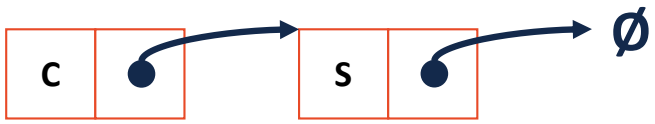
Linked Memory

List.h

```
28 class ListNode {
29   T & data;
30   ListNode * next;
31   ListNode(T & data) : data(data), next(NULL) { }
32 };
```

Linked Memory

Linked Memory



List.h

```
 1  #pragma once
 2
 3  template <class T>
 4  class List {
 5    public:
…      /* ... */
28    private:
29      class ListNode {
30        T & data;
31        ListNode * next;
32        ListNode(T & data) :
           data(data), next(NULL) { }
33      };
34
35
36
37
38
39  };
40
41
```

List.hpp

```
 1  #include "List.h"
 2
 3  template <class T>
 4  void List<T>::insertAtFront(const T& t) {
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22  }
```

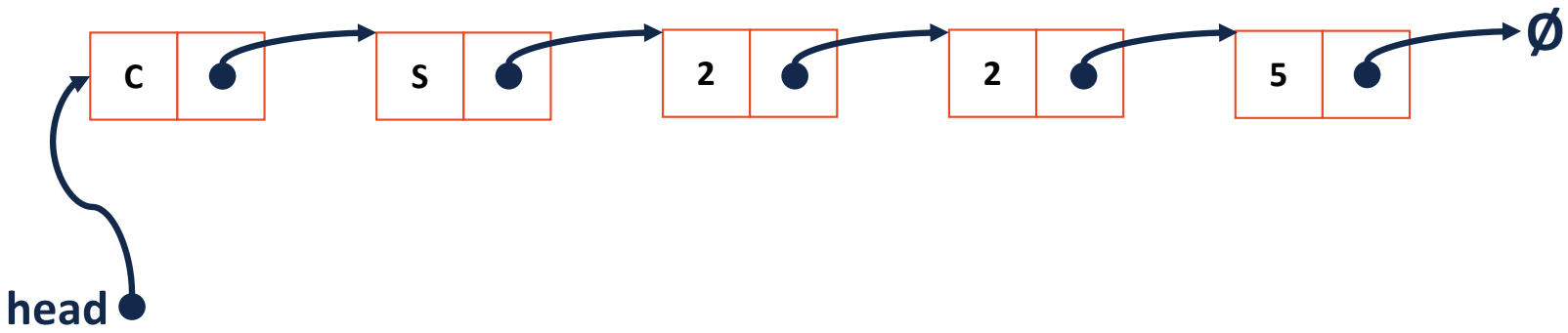
Running Time of Linked List `insertAtFront`

```
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

List.cpp

```
14  void List<T>::printReverse()
    const {
15
16
17
18
19
20
21
22  }
```

Linked Memory



Running Time of Linked List `printReverse`

List.cpp

```
24 template <typename T>
25 T List<T>::operator[](unsigned index) {
26
27
28
29
30
31 }
```

List.cpp

```
33 ListNode *& List<T>::_index(int index) const {
34
35
36
37
38
39
40 }
```