CS 225

Data Structures

Feb. 4 – Templates and Linked Memory

Wade Fagen-Ulmschneider, Craig Zilles

animalShelter.cpp

```
5  class AnimalShelter {
6    public:
7      Animal & adopt();
…      // ...
   };
```

animalShelter.cpp

```
 1 class Animal {
 2   public:
 3     void speak() {                                    }
 4 };
 5
 6 class Dog : public Animal {
 7   public:
 8      void speak() {                                     }
 9 };
10
11 class Cat : public Animal {
12   public:
13      void speak() {                                     }
14 };
```

Abstract Class:

[Requirement]:

[Syntax]:

[As a result]:

virtual-dtor.cpp

```
 4 class Cube {
 5   public:
 6     ~Cube() { std::cout << "~Cube() invoked."
                           << std::endl; }
 7 };
 8
 9 class RubikCube : public Cube {
10   public:
11     ~RubikCube() { std::cout << "~RubikCube() invoked."
                                << std::endl; }
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() { std::cout << "~CubeV() invoked."
18                             << std::endl; }
19 };
20
21 class RubikCubeV : public CubeV {
22   public:
23     ~RubikCubeV() { std::cout << "~RubikCubeV() invoked."
24                                 << std::endl; }
25 };
```

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

```
waf@siebl-2215-02:/mnt/c/Users/waf/Desktop/cs225/_lecture/09-linkedMemory$ make
clang++  virtual-dtor.cpp -lm -o virtual-dtor
waf@siebl-2215-02:/mnt/c/Users/waf/Desktop/cs225/_lecture/09-linkedMemory$ ./virtual-dtor
Non-virtual dtor:
~Cube() invoked.
Virtual dtor:
~RubikCubeV() invoked.
~CubeV() invoked.
waf@siebl-2215-02:/mnt/c/Users/waf/Desktop/cs225/_lecture/09-linkedMemory$
```

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.cpp

```
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

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.cpp

```
 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  }
```

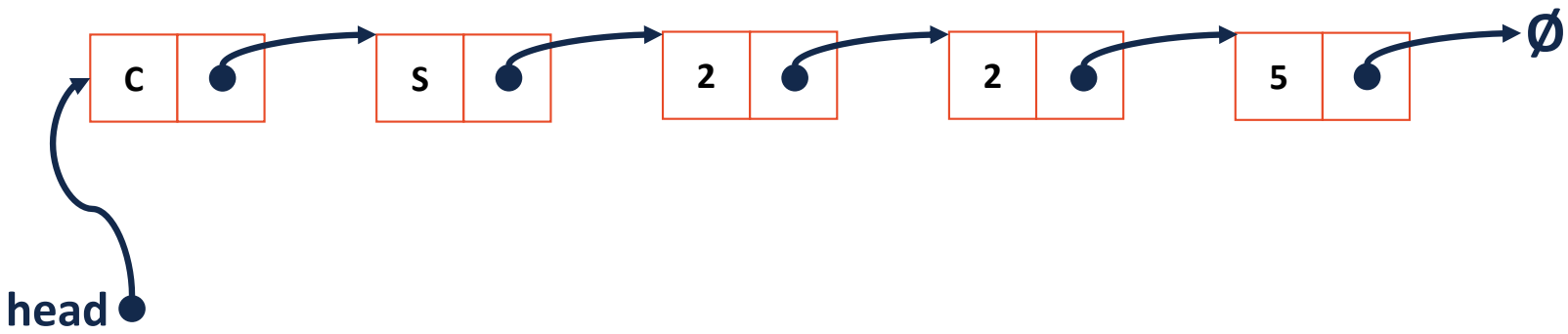
Running Time of Linked List `insertAtFront`

```
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`