|  |  |
| --- | --- |
|  |  |
| 자료구조와C++프로그래밍 |  |
|  |  |
|  | **이름 : 장혁수****학과 : 소프트웨어학과****학번 : 2019125055****수업시간 : 월 10:30, 수09:00** |
|  | **과제번호 : 과제 4** |

### 문제 정의

이번 과제4는 전의 과제3과 많은 부분에서 비슷하다.

즉, 원하는 카드 개수만큼 카드를 받고 정렬하는데, 미리 정해진 카드의 우선순위에 따라 정렬하는 과정을 출력하는 문제이다.

하지만 과제3과 과제4가 다른 점은 과제3에서는 배열에 데이터들을 저장했지만, 이번 과제4에서는 연결 리스트에 데이터들을 저장해야 한다는 것이다.

이에 과제4에서 요구하는 주요 중점사항은 아래와 같다.

1. 연결 리스트를 구현한다.
2. 연결 리스트를 위한 반복자를 구현한다.
3. 카드라는 클래스를 만들고 위에서 구현한 연결 리스트와 반복자를 활용해서 카드 하나하나를 리스트 앞에 push한다.
4. 리스트에서 카드들을 정렬할 때 우선순위에 따라 정렬해야 한다. 이에 적절한 위치에 배열될 때까지 반복자를 통해 바로 다음 카드와 swap한다.
5. 카드가 들어가고 위치가 변경되는 과정을 계속해서 출력으로 보여준다.

이에 근거해서 문제 해결을 의한 과정을 대략 생각해보았다.

일단template 클래스 Chain와 ChainNode를 만들어서 전형적인 연결 리스트를 구현하고, Chain클래스 내부에 iterator를 구현하고자 한다.

그리고 Chain<Card>타입의 객체를 멤버로 가지는 Deck 클래스를 또한 구현해서 Card 객체들을 연결리스트에 저장하려고 한다. 이때, Deck 클래스에 Card객체를 적절히 배열하는 과정을 출력하는 멤버 함수를 구현하고자 한다.

이후 main함수에서 Deck 클래스 타입의 변수를 선언하고 카드 객체들을 중복없이 랜덤하게 만든 후 구현한Deck의 멤버함수를 이용해 만든 카드 객체를 연결 리스트에 적절히 집어넣겠다.

### 문제 해결

◎ Chain, ChainNode 클래스를 통해 연결 리스트 구현하기

먼저, 연결 리스트란 배열과 비슷하게 노드(원소)들을 리스트에 저장한다. 하지만 배열과 다르게 노드(원소)들의 저장 순서가 리스트의 표현 순서와 같지 않다. 즉, 각 리스트 노드에는 다음 노드를 가리키는 포인터가 있고 이 포인터를 이용해서 체인 형태로 리스트를 표현한다.

여기서 원형 연결 리스트 등 다양한 형태의 연결 리스트가 있는데 여기서 단순 연결 리스트를 사용하고자 한다.

즉, 정리하자면Chian과 ChainNode라는 두 클래스를 조합해서 단순 연결 리스트를 구현하겠는데 여기서 Chain은 리스트의 첫 번째 노드를 지시하는 포인터, first만 포함하도록 정의하고 ChainNode 클래스는 리스트의 각 노드를 나타낸다. 추가로 구현한 것은 어떤 타입의 데이터도 저장할 수 있게 하고자 Chain과 ChainNode를 템플릿 클래스로 선언했다는 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이에 먼저 ChainNode 클래스를 위와 같이 구현해 보았다.

여기서 핵심은 Chain 클래스에서 ChainNode 클래스를 쓸 수 있게 ChainNode안에서 Chain을 friend라고 선언한 것과 각각의 노드에는 T타입의 데이터와 연결 리스트이 다음 노드를 가지는 포인터 변수를 멤버로 선언했다는 것이다.

다음으로 Chain클래스를 다음과 같이 구성했다. 이때 Chain 클래스 내 반복자는 이어지는 뒤의 내용에서 다룰 예정이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

요점은 리스트의 첫 번째 노드를 가리키는 포인터 변수 frist가 멤버로 있다는 것이다. 이 first변수는 Chain 클래스가 생성되면 생성자에 의해 먼저 first는 0의 값으로 초기화 된다.



그리고 메모리의 관리를 위해 freeAll이라는 함수를 따로 구현했고, 이 함수는 파괴자가 호출될 때 실행되도록 구현했다. freeAll함수의 코드는 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

또한 핵심적으로 데이터 e를 가지는 노드를 동적 할당하고 리스트의 맨 앞에 push하는 InsertFront함수 또한 구현했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

마지막으로 데이터 e를 가지고 있는 노드를 찾은 후, 찾은 노드와 다음 노드를 서로 swap해주는 함수 NodeSwap함수를 구현했다.

여기서 생각해 볼 수 있는 경우가 먼저 리스트에 안에 데이터 e를 가지고 있는 노드가 있는 경우와 없는 경우이다. 이때 찾고자 하는 노드가 있으면 정상적으로 swap하고 없다면 에러 메시지를 출력하고 함수를 종료하게 했다.

이후 찾고자 하는 노드가 있는데 이 노드가 있는 위치에 따라 또 경우가 나뉜다.

즉, 1. 맨 앞에 있는 경우, 2. 맨 뒤에 있는 경우, 3. 나머지 경우이다.

1의 경우에는 찾고자 하는 노드의 주소 값만 알면 적절히 swap할 수 있고, 2의 경우는 그냥 아무 행동없이 함수를 종료하면 된다. 여기서 문제는 나머지 경우이다. 이 때는 찾고자 하는 노드의 주소 값만 알면 되는 게 아니라 그 전의 노드의 주소 값도 알아야 한다.

위에서 언급한 경우들을 생각하면서 다음과 같이 코드를 구현했다.

텍스트, 벽, 스크린샷, 은색이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

즉, 찾고자 하는 노드의 주소 값을 저장할 current 변수와 이 current를 따라가는 previous 변수 또한 적절히 선언했다. 이후 while 문을 통해 찾고자 하는 노드를 찾았거나 연결 리스트 끝까지 간 경우 종료되도록 만들었다. 그리고 처음에 찾고자 한 노드가 있는지 없는지 검사해서 없다면 함수가 종료되게 했다. 마지막으로 찾고자 하는 노드가 리스트 어디에 있는지 조건문을 활용해 구별하고 경우에 맞추어서 swap을 하거나 안 하거나 했다.

◎ 연결 리스트를 위한 반복자(iterator) 구현하기

연결 리스트에서 반복자를 쓰는 이유는 객체의 원소들에 대해 체계적인 접근을 하기 위해서이다.

즉, 리스트의 순회에 필요한 모듈화된 함수를 제공함으로 프로그램에서 단순히 함수를 호출함으로 다양한 작업(노드 간의 포인터 이동, 첫 노드인지 마지막 노드인지 등)을 할 수 있게 하기 위해서이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 화면, 검은색이(가) 표시된 사진

자동 생성된 설명

이에 위와 같이Chain 클래스 내부에서 반복자 iterator를 구현했다. 즉, iterator는 Chain의 공용 중첩 멤버 클래스이다.

그리고 마지막 두 함수는 current값을 first, null로 가지는 반복자를 반환하는 공용 멤버 함수이다

◎ Card 클래스와 Deck 클래스를 이용해 우선순위에 따라 카드들 정렬하기

먼저 연결 리스트에서 각 노드의 데이터가 될 Card 클래스를 다음과 같이 선언했다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

주요 부분으로는 어떤 카드인지 알기 위해 suit, rank라는 변수를 통해 정보를 저장하고 있다. 또한, 나중에 카드를 출력할 때 이용하고자 정적 데이터 멤버를 하나 만들었다는 것이다.

그리고 이후 Chain 클래스에서 반복자를 편하게 사용할 수 있도록 Card 클래스 내부에 연산자 >, != 에 대한 오버로딩을 구현했다는 점도 있다.

다음으로 핵심이 되는 Deck 클래스를 살펴보자

텍스트이(가) 표시된 사진

자동 생성된 설명

코드에서 볼 수 있듯이 Deck 클래스는 Chain<Card> 타입의 deck 변수를 데이터 멤버로 가지고 있다. 즉, deck은 클래스 Card 타입의 데이터를 가지고 있는 노드들을 체인 형식으로 가지고 있는 연결 리스트의 첫 번째 노드의 주소 값을 가지고 있을 것이다.

이후 공용 멤버 함수로는 간단하게 Deck이 가리키고 있는 연결 리스트의 모든 Card 데이터들을 출력하는 showDeck함수와 매개변수 Card c를 연결 리스트에 적절히 정렬하는 과정을 보여주는 PushCard 함수가 있다.

먼저 showDeck 함수을 살펴보겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명

연결 리스트의 모든 Card 데이터들을 출력하기 위해선 모든 노드에 접근해야 한다. 이에 앞서 구현한 반복자를 통해 보다 깔끔하게 접근하려고 한다.

즉, Chain<Card>에 해당하는 반복자 ait을 이용해 모든 노드에 접근하는데 그때마다 가리키는 노드의 Card 데이터의 ShowCard함수를 호출함으로 그 노드의 Card의 정보를 출력하게 했다.

마지막으로 Deck 클래스의 PushCard 함수를 보겠다.

앞서 문제의 정의에서 보다시피 Card 데이터를 가지고 있는 노드를 연결 리스트에 정렬할 때, 일단 리스트의 맨 앞부분에 push한다. 그리고 연결된 다음 노드가 존재한다면 그 노드의 Card 데이터의 우선순위를 서로 비교해서 새로 push한 노드의 데이터의 우선순위가 높다면 서로 swap하고 이렇게 계속 swap을 반복하다가 다음 노드의 데이터의 우선순위가 더 높을 때 swap을 멈춤으로 정렬을 마친다.

이 과정을 생각하면서 아래의 코드를 보겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저, Deck::showDeck함수를 통해 연결 리스트의 모든 Card 데이터를 출력해준다.

이후 Chain<Card>::InsertFront 함수를 이용해 Card 객체 C를 연결 리스트 맨 앞에 push을 하고 또다시 showDeck함수를 이용해 연결 리스트를 출력한다.

그리고 연결 리스트의 노드들을 순회하기에 Chain<Card>에 해당하는 반복자 ait, bit을 선언했다. 이때 ait은 연결 리스트의 첫 번째 노드를, bit은 연결 리스트의 두 번째 노드를 가리키고 있다. 즉, ait은 새롭게 push한 Card 데이터를 가지고 있는 노드를 가리키고 bit은 ait이 가리키는 노드의 다음 노드를 가리키고 있다.

이후 while 반복문을 통해 bit이 아무것도 안 가리키고 있거나(ait이 마지막 노드를 가리키는 경우) ait이 가리키는 노드의 Card 데이터의 우선순위가 bit이 가리키는 노드의 Card 데이터의 우선순위보다 작은 경우일 때까지 반복하게 했다.

그리고 반복문 내부를 보면 일단 Chain<Card>::NodeSwap함수를 이용해 ait이 가리키는 노드와 bit이 가리키는 노드를 서로 swap하고 연결 리스트를 출력했고, bit이 다른 노드를 가리키게 바꾸어 주었다. 여기서 핵심은 ++(++bit) 연산을 통해 bit이 다른 노드를 가리키게 했다는 것이다. 왜냐하면NodeSwap함수로 인해서 연결 리스트의 노드들의 순서가 바뀌기는 했지만 정작 bit은 여전히 같은 노드를 가리키고 있다. 즉, bit은 ait이 가리키는 노드의 다음 노드를 가리켜야 하기에 ++(++bit)연산을 통해 bit이 가리키는 노드를 변경시켜 주었다. 참고로 ait은 여전히 새롭게 push한 Card 데이터를 가지고 있는 노드를 가리키기에 값을 변경시키지는 않았다.

위의 내용을 아래 그림을 통해 간단히 표현해 보았다.

* NodeSwap함수 호출 전 연결 리스트

ait

bit

* NodeSwap함수 호출 후 연결 리스트

bit

ait

* ++(++bit)연산 후 연결 리스트

bit

ait

◎ main함수에서 랜덤하게 카드를 만들고 구현한 클래스들 활용하기

이제 필요한 클래스와 함수들은 다 구현했으니 main에서 카드들을 중복없이 랜덤하게 받고, 구현한 클래스와 함수를 적절히 사용하면 되겠다.

먼저 카드의 중복을 없애기 위해 4x13의 2차원 bool형 배열을 선언했고 이 배열의 모든 값은 처음에 false로 초기화했다. 그리고 이후 카드를 deck에 집어넣는다면 그 카드를 나타내는 배열의 공간의 값을 1로 바꾸어서 구별하도록 했다.

예를 들어서 suit = 2, rank = 1인 카드가 들어왔다면, bool in\_hand[4][13] 배열의 in\_hand[2][1] 값을 true로 바꾸어 주는 것이다.

그리고 Card 데이터들을 저장할 연결 리스트를 가리키는 클래스 Deck 의 포인터 변수 MyDeck을 선언해 Deck 객체를 동적할당 했다.

다음으로 랜덤하게 받기 위해 필요한 함수인 srand, rand, time 함수를 쓰기 위해 헤더파일 stdlib.h와 time.h를 include 시켰다.

랜덤하게 받는 원리는 rand함수를 통해 값을 랜덤하게 받는 것이다. 이때, srand함수의 매개변수로 들어오는 seed 값에 따라 rand함수의 결과가 정해진다. 이에 seed 값으로 time함수를 이용해 항상 변하게 했다.

위의 내용을 토대로 main함수에 일단 다음과 같이 코드를 구현했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이후 반복문을 통해 #define으로 정의된 상수 NUM\_CARDS만큼의 카드들을 만들 때까지 반복하게 해주었다. 그리고 이 반복문 안에서 핵심적으로 선택된 정보에 맞추어서 새로운 Card 객체를 생성 및 출력했고, 또한 Deck::PushCard 함수를 이용해 연결 리스트에 만들어진 카드 객체를 push하고 정렬하는 과정을 출력했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이후 원하는 카드의 개수를 5로 설정하고 코드를 실행시킨 결과 아래와 같다.

테이블이(가) 표시된 사진

자동 생성된 설명

### 결론

연결 리스트가 배열과 비교해서 가지는 하나의 가치는 각각의 원소(노드)들이 하나의 공간에 저장되는 게 아니라 체인과 같은 형태도 서로 연결되어서 저장된다는 것이다. 왜냐하면 배열은 중간에 원소를 push하거나 pop할 때 메모리 공간을 절약하기 위해 다소 귀찮은 연산을 해야 하지만 연결 리스트는 보다 간단하게 구현할 수 있기 때문이다. 하지만 분명 연결 리스트에도 그만큼의 단점이 있고 앞으로 상황에 맞추어서 배열과 연결 리스트를 적절히 사용해야 할 것이다.

문제를 풀면서 힘들었던 점은 Chain클래스와 ChainNode 클래스를 템플릿 클래스로 구현한 것이었다. 즉, Chain, ChainNode 클래스에 신경을 주로 써야 했지만, 또한 Card, Deck 클래스에서도 이에 맞추어서 신경을 써야 했기에 혼란이 조금 있었던 것 같다. 그래도 범용성을 위해서는 꼭 필요한 작업인 것 같고 연습을 한 좋은 시간이었다.