3주차 결과보고서

전공: 신문방송학과 학년: 3학년 학번: 20191150 이름: 전현길

1. 실험 시간에 작성한 프로그램의 알고리즘과 자료구조를 요약해 기술하시오.

클래스와 파생 클래스 개념을 바탕으로 입력된 크기만큼 int형 변수를 저장하는 Array 배열, 입력된 숫자의 범위만큼 int형 변수를 저장하는 RangeArray 배열을 구현한다. 이 때, RangeArray 배열은 음수가 입력될 수 있다.

1-1) Array 구현

생성 시 배열의 크기를 입력받고, new int[size]로 메모리를 할당한다. 소멸 시 할당한 메모리를 해제한다. 따라서 요구되는 멤버 변수로 int len, int* data 변수가 필요하다. 기능으로는 길이(=len) 반환, 원소 삽입, 원소값 반환, 배열 내용 출력을 제공한다.

배열 내용 출력은 반복문을 이용해 data를 len만큼 출력한다. 원소 삽입, 원소값 반환의 간편화를 위해 연산자 오버로딩을 활용한다. 원소 삽입을 위해서 요구된 idx 값의 변수를 참조하는 int& 변수를 반환하는 int& operator[] 함수를 구현하고, 원소 출력을 위해서 요구된 idx의 변수를 반환하는 int operator[] const 함수를 구현한다.

생성 시 음수 크기의 메모리를 할당할 경우, 배열의 범위 밖에서 원소 삽입/원소값 반환을 요구할 경우 오류 메시지를 출력하고 오류값을 반환하여 에러 처리한다.

1-2) RangeArray 구현

Array 함수의 파생 클래스로 자료구조와 알고리즘이 거의 유사하지만, 음수를 허용하는 범위값을 입력받는다. 따라서 요구되는 멤버 변수로 시작 idx 값 low, 끝 idx 값 high가 추가된다. 생성 시 RangeArray(int i, int j) : Array(j - i + 1) { … } 코드로 Array 클래스의 size 변수를 저장한다. Array 기반 클래스에서 low값, high값을 각각 반환하는 기능 baseValue(), endValue()가 추가된다.

연산자 오버로딩을 구현하기 위해서 Array 클래스의 operator[] 함수를 불러와서 사용하되, idx값을 맞추기 위해 사용자에게 입력받은 idx를 i - low 처리한다.

에러 처리는 Array의 에러 처리 방식을 재활용할 수 있다.

2. 과제 문제를 해결하기 위한 알고리즘 및 자료구조를 요약하여 기술하시오.

문자열을 복사하고, 저장할 수 있는 간단한 문자열 클래스 Str을 구현한다. **멤버 변수**로 문자열의 길이를 저장하는 int len 변수, 1차원 문자열의 시작 주소를 저장하는 char* str을 갖는다. 가지고 있는 기능은 다음과 같다.

1. 길이 반환
2. 내용 반환
3. (char*/Str 클래스) 문자열 비교
4. (char*/Str 클래스) 문자열 복사

생성 시 Str(문자열 길이), Str(문자열 내용) 형태로 사용자 입력을 받으며, 입력받은 길이나 문자열의 길이에 따라 메모리를 할당한 뒤 내용이 입력되었을 경우 strcpy() 함수로 내용을 복사한다. 소멸 시 delete[] str로 메모리 할당을 해제한다.

기능 1, 2를 위해 길이 변수 len을 반환하는 int length() 함수, 문자열의 주소를 반환하는 char* contents() 함수를 구현한다. 기능 3을 위해서 char* a 변수를 인자로 받을 경우 strcmp(str, a)를 반환하는 함수, Str& a 변수를 인자로 받을 경우 strcmp(str, a.str)을 반환하는 함수를 구현한다.

기능 4를 위해서 연산자 오버로딩 함수 void operator=(char* a), void operator=(Str& a)를 구현한다. 두 경우 모두 strlen(a) 또는 a.len을 통해 인자의 문자열 길이를 확인한 뒤, 멤버 변수 len에 저장한다. 이후 멤버 변수 str의 메모리를 해제한 후 len + 1만큼 재할당하고, strcpy() 함수를 활용해 내용을 복사한다.

일반적으로는 문자열의 내용이 현재 할당된 메모리보다 작을 경우 메모리를 해제한 뒤 재할당하는 대신 내용을 복사한 뒤 끝에 널 문자만 적지만, 현재 구현한 Str 클래스의 경우 바뀐 문자열의 길이와 관계없이 무조건 메모리를 해제한 후 재할당한다. 이 경우 문자열을 조금 갱신할 때마다 메모리 해제 후 재할당이 이뤄지므로 오버헤드가 커지지만, 메모리 낭비를 최소화할 수 있다는 장점이 있다.