**자료구조 리스트 문제**

리스트 성공 코드

202112936 엄정석

**A. 다항식 프로그램(희소 다항식)**

-문제

|  |
| --- |
| **다항식 프로그램**  **p(x)=10x100+6과 같이 최고차항의 차수가 매우 크고 대부분 항의 계수가 0인 다항식을 희소 다항식이라 한다. 이러한 다항식을 구현하기 위해 배열을 사용하면 메모리의 낭비가 심하다. 따라서 희소 다항식은 연결 리스트를 이용하여 구현하는 것이 좋다. 다음과 같이 처리되도록 프로그램 2.7을 참고하여 연결 리스트를 이용한 희소 다항식 프로그램을 구현하라.**    **Input**  **입력에는 두 개의 식이 순서대로 입력된다. 첫 번째 식은 첫 줄에는 항의 개수 n (1<=n<=20)이 들어온다.   다음 줄 부터는 최고차항 순서로 (계수, 차수)가 들어오며 공백으로 구분된다. 두 번째 식도 같은 방법으로 입력된다.**    **Output**  **먼저, 두 개의 식을 “(항의 개수) = 다항식”의 형식으로 각 줄에 출력한다. 이때 Sample Output에 맞추어 공백을 구분하는데 유의한다. 각 계수는 소수점 첫째자리 까지 출력하고 계수가 0인 항은 출력하지 않는다 (Sample Output 참고). 그리고 마지막줄에는 두 개의 다항식을 더하여 하나의 다항식으로 출력한다.**    **Sample 1**  **Input**  **3 3 12 2 8 1 0 4 15 11 14 3 2 2 1 1**  **Output**  **(3) = 3.0 x^12 + 2.0 x^8 + 1.0 x^0 (4) = 15.0 x^11 + 14.0 x^3 + 2.0 x^2 + 1.0 x^1 (7) = 3.0 x^12 + 15.0 x^11 + 2.0 x^8 + 14.0 x^3 + 2.0 x^2 + 1.0 x^1 + 1.0 x^0**  **Sample 2**  **Input**  **4 3 12 2 8 10 3 1 0 3 15 12 -14 8 2 3**  **Output**  **(4) = 3.0 x^12 + 2.0 x^8 + 10.0 x^3 + 1.0 x^0 (3) = 15.0 x^12 + -14.0 x^8 + 2.0 x^3 (4) = 18.0 x^12 + -12.0 x^8 + 12.0 x^3 + 1.0 x^0**    **Sample 3**  **Input**  **4 10 5 32 4 2 3 5 2 5 1 12 2 10 3 5 4 3 5 2**  **Output**  **(4) = 10.0 x^5 + 32.0 x^4 + 2.0 x^3 + 5.0 x^2 (5) = 1.0 x^12 + 2.0 x^10 + 3.0 x^5 + 4.0 x^3 + 5.0 x^2 (6) = 1.0 x^12 + 2.0 x^10 + 13.0 x^5 + 32.0 x^4 + 6.0 x^3 + 10.0 x^2** |

-성공 코드(C++)

|  |
| --- |
| /\*  p(x)=10x100+6과 같이 최고차항의 차수가 매우 크고 대부분 항의 계수가 0인 다항식을 희소 다항식이라 한다.  이러한 다항식을 구현하기 위해 배열을 사용하면 메모리의 낭비가 심하다.  따라서 희소 다항식은 연결 리스트를 이용하여 구현하는 것이 좋다.  다음과 같이 처리되도록 프로그램 2.7을 참고하여 연결 리스트를 이용한 희소 다항식 프로그램을 구현하라.  (고쳐야 하는 것: 출력 형식, 마지막 항의 '+' 삭제, 그리고 계수를 실수형 그대로 출력하기)  \*/  #include <cstdio>  #include <cstdlib>  #include <iostream>  using namespace std;  class Node  {  private:  float coef; //계수  int exp; //지수  Node\* link;  public:  Node(float c = 0, int e = 0) :coef(c), exp(e), link(NULL) { }  Node\* getLink(){  return link;  }  void setLink(Node\* next){  link = next;  }  void display(){  printf("%.1f x^%d", coef, exp);  }  //자신의 다음에 새로운 노드 n을 삽입하는 함수  void insertNext(Node\* n){  if (n != NULL){  n->link = link;  link = n;  }  }  //자신의 다음 노드를 리스트에서 삭제하는 함수  Node\* removeNext()  {  Node\* removed = link;  if (removed != NULL)  link = removed->link;  return removed;  }  friend class polyList;  };  //다항식 링크 리스트  class polyList{  private:  Node org;  public:  polyList() : org(0) { }  ~polyList() { clear(); }  void clear(){  while (!isEmpty())  delete remove(0);  }  Node\* getHead(){  return org.getLink();  }  bool isEmpty(){  return getHead() == NULL;  }  //pos번째 항목을 반환함  Node\* getEntry(int pos){  Node\* n = &org;  for (int i = -1; i < pos; i++, n = n->getLink())  if (n == NULL)  break;  return n;  }  //리스트의 어떤 위치에 항목 삽입  void insert(int pos, Node\* n)  {  Node\* prev = getEntry(pos - 1);  if (prev != NULL)  prev->insertNext(n);  }  //리스트의 어떤 위치의 항목 삭제  Node\* remove(int pos){  Node\* prev = getEntry(pos - 1);  return prev->removeNext();  }  //리스트의 항목 개수 반환  int size(){  int count = 0;  for (Node\* p = getHead(); p != NULL; p = p->getLink())  count++;  return count;  }  void Add(polyList\* that){  polyList temp;  int count = 0;  Node\* p = getHead();  Node\* t = that->getHead();  while (p != NULL && t != NULL){  if (p->exp == t->exp){  temp.insert(count++, new Node(p->coef + t->coef, p->exp));  p = p->getLink();  t = t->getLink();  }  else if (p->exp > t->exp){  temp.insert(count++, new Node(p->coef, p->exp));  p = p->getLink();  }  else{  temp.insert(count++, new Node(t->coef, t->exp));  t = t->getLink();  }  }  for (; p != NULL; p = p->getLink())  temp.insert(count++, new Node(p->coef, p->exp));  for (; t != NULL; t = t->getLink())  temp.insert(count++, new Node(t->coef, t->exp));  //cout << "A+B =(" << temp.size() << "항)= ";  cout << "(" << temp.size() << ") = ";  temp.display();  }  void input(){  int count, e;  float c;  //cout << "희소 다항식의 항의 개수를 입력하세요: ";  cin >> count;  //cout << "각 항의 계수와 지수 입력(최고차항부터 " << count << "개)" << endl;  for (int i = 0; i < count; i++){  //cout << i + 1 << "번째 항 : 계수 지수 = ";  cin >> c >> e;  insert(i, new Node(c, e));  }  }  //출력  void display(){  int count = 1;  if (size() == 0){  cout << "다항식이 존재하지 않습니다" << endl;  return;  }  else{  for (Node\* p = getHead(); p != NULL; p = p->getLink()){  p->display();  if (p->getLink() != NULL) {  cout << " + ";  } //이에 따라 마지막 항은 '+'이 출력되지 않는다!  }  cout << endl;  }  }  };  int main(void)  {  //ios\_base::sync\_with\_stdio(false);  //cin.tie(NULL); cout.tie(NULL);  //실행 속도 증가  polyList A, B, C;  A.input();  B.input();  //cout << "A=(" << A.size() << "항) = ";  cout << "(" << A.size() << ") = ";  A.display();  //cout << "B=(" << B.size() << "항) = ";  cout << "(" << B.size() << ") = ";  B.display();    A.Add(&B);  return 0;  } |

-난점: 매크로는 왜 쓸 수 없는가?

**B. 라인편집기**

-문제

|  |
| --- |
| ****라인 편집기****   연결리스트를 이용한 라인편집기를 구현하고자 한다. 편집기의 기능은 한 라인 삽입, 삭제, 변경, 찾아 바꾸기, 출력 기능이 제공된다. 각 기능에 대한 세부 설명은 다음과 같다:   * 한 라인 삽입(i): 행번호, 문자열을 입력하면, 그 행에 한 라인을 삽입 * 한 라인 삭제(d): 행번호를 입력하면, 그 행을 삭제 * 한 라인 변경(r): 행번호와 문자열을 입력하면, 그 행의 내용을 변경 * 찾아 바꾸기(f): 첫 줄에는 찾을 문자열을 입력, 두번째 줄에는 바꿀 문자열을 입력 받아서, 해당 문자열을 찾아서 바꿀 문자열로 변경 (해당 문자열을 모두 찾아서 바꿈. 문자열의 부분 일치가 아니라, 라인 전체 일치에 대해서 처리함) * 출력(p): 편집기의 모든 내용을 라인 번호와 함께 출력 * 종료(q): 프로그램을 종료    ****Input**** 라인편집기 처리할 기능을 입력 받아서 처리한다. 종료할 때까지 반복 실행한다.  단, 삽입되거나 변경될 문자열의 길이는 20을 넘지 않는다. 문자열 사이에 공백이 있을 수 있음.  라인편집기의 입력에는 오류가 발생하지 않는다고 가정한다.    참고: 문자를 입력 받은후 행바꿈문자 처리를 위해 버퍼를 비워주는 fflush(stdin)가 리트머스에서 제대로 적용되지 않으면 다음 코드를 참고하셔도 됩니다.  char line[MAX\_CHAR\_PER\_LINE];  while (getchar() != '\n');    fgets(line, MAX\_CHAR\_PER\_LINE, stdin);   ****Ouput****       출력 커맨드(p)를 입력 받을 때, 해당내용들을 출력한다. 마지막에 “EOF” 를 출력한다.        출력형식: <행번호 한라인내용>   ****Sample Example 1:**** **Input**  **i**  0  Contents  **i**  1  Basic  **i**  2  Stack, Queue, Deque  **i**  3  Linked List  **p                            // 첫번째 출력 명령**  **d**  2  **p                            // 두번째 출력 명령**  **r**  2  Array  **p                            // 세번째 출력 명령**  **f**  Contents  Contents of Table  **p                            // 네번째 출력 명령**  **q**   ****Output**** 0: Contents                   **// 첫번째 p 출력 결과**  1: Basic  2: Stack, Queue, Deque  3: Linked List  EOF  0: Contents                    **// 두번째 p 출력 결과**  1: Basic  2: Linked List  EOF  0: Contents                   **// 세번째 p 출력 결과**  1: Basic  2: Array  EOF  0: Contents of Table           **// 네번째 p 출력 결과**  1: Basic  2: Array  EOF |

-성공 코드(C++)

|  |
| --- |
| #pragma once  #include <cstdio>  #include <stdio.h>  #include <cstring>  #include <string.h>  #include <cstdlib>  #include <iostream>  #include <string>  using namespace std;  #define MAX\_CHAR\_PER\_LINE 1000  #pragma warning(disable:4996)  string editorcopy[MAX\_CHAR\_PER\_LINE]; //라인에 입력된 내용 저장하는 문자열 배열  int stringcount = 0; //카운팅 변수  class Line {  char data[MAX\_CHAR\_PER\_LINE]; //개행당 1000자까지 저장(데이터 필드)  public:  Line(const char\* line = " ") { strcpy(data, line); }  //출력  void print(FILE\* fp = stdout, int i = 0) {  //fprintf(fp, "%s", data);  editorcopy[i].append(data);  }  //비교  bool hasData(char\* str) { return strcmp(str, data) == 0; }  //(1) str1 < data 인 경우에는 음수 반환  //(2) str1 > data 인 경우에는 양수 반환  //(3) str == data 인 경우에는 0을 반환 합니다.  };  class Node : public Line {  Node\* link; //링크 필드  public:  Node(const char\* str = " ") : Line(str), link(NULL) { }  void setLink(Node\* p) { link = p; } //다음 링크 설정  Node\* getLink() { return link; } //링크 값 반환  void insertNext(Node\* p) { //자신 다음에 p 추가  if (p != NULL) {  p->link = link;  link = p;  }  }  Node\* removeNext() { //자신 다음의 요소 삭제  Node\* removed = link;  if (removed != NULL) {  link = removed->link;  }  return removed;  }  };  class LinkedList {  Node org; //헤드 노드(헤드 포인터 X)  public:  LinkedList() : org("") { }  ~LinkedList() { clear(); }  LinkedList(const LinkedList& origin) : org(origin.org) { }  //새롭게 메모리 할당  void clear() { while (!isEmpty()) delete remove(0); }  Node\* getHead() { return org.getLink(); } // 헤드 노드의 값  bool isEmpty() { return getHead() == NULL; }  //pos번째 항목 반환  Node\* getEntry(int pos) {  Node\* n = &org;  for (int i = -1; i < pos; i++, n = n->getLink()) {  //-1부터 시작하는 이유: 헤드 노드의 주소가 반환되도록 설계  if (n == NULL) break;  }  return n;  }  //리스트의 pos 위치에 항목 추가  void insert(int pos, Node\* n) {  Node\* prev = getEntry(pos - 1);  if (prev != NULL) {  prev->insertNext(n);  }  }  //리스트의 pos 위치에 항목 삭제  Node\* remove(int pos) {  Node\* prev = getEntry(pos - 1);  return prev->removeNext();  }  //탐색  Node\* find(char\* val) {  for (Node\* p = getHead(); p != NULL; p = p->getLink()) {  if (p->hasData(val)) return p;  }  return NULL;  }  //pos번째 노드를 다른 노드로 교체  void replace(int pos, Node\* n) {  Node\* prev = getEntry(pos - 1);  if (prev != NULL) {  delete prev->removeNext();  prev->insertNext(n);  }  }  //pos번째 val을 find + replace 후 n 반환(자료구조에서 새롭게 요구하는 함수)  Node\* find\_and\_replace\_number\_return(int pos, char\* val, Node\* n) {  for (Node\* p = getHead(); p != NULL; p = p->getLink()) {  if (p->hasData(val) == true) {  Node\* prev = getEntry(pos - 1);  if (prev != NULL) {  delete prev->removeNext();  prev->insertNext(n);  return prev;  }  }  }  return NULL;  }  //val을 find + replace 후 n 반환  Node\* find\_and\_replace\_number\_return2(char\* val, Node\* n) {  Node\* p = getHead(); //헤드 노드  Node\* prev = &org; //헤드 노드의 주소를 가리킴  while (p != NULL) {  if (p->hasData(val)) { //val 값이 있을 때  prev->setLink(n); //prev는 n을 가리킨다.  n->setLink(p->getLink()); // n은 p의 다음 링크를 가리킨다.  delete p; // p를 삭제한다.  return n;  }  prev = p;  p = p->getLink(); //다음 링크값을 따란다.  }  return NULL;  }  //리스트 항목 개수 반환  int size() {  int count = 0;  for (Node\* p = getHead(); p != NULL; p = p->getLink()) {  count++;  }  return count;  }  };  class LineEditor : public LinkedList {  public:    LineEditor() { }  ~LineEditor() { }    //새롭게 메모리 할당  LineEditor(const LineEditor& origin) : LinkedList(origin) { }  //출력  void Display(FILE\* fp = stdout) {  int i = 0;  for (Node\* p = getHead(); p != NULL; p = p->getLink(), i++) {  //fprintf(stderr, "%3d ", i);  p->print(fp, stringcount);  stringcount++;  }  editorcopy[stringcount] = "EOF";  stringcount++;  }  //한 라인 입력 처리  void InsertLine() {  int position;  char line[MAX\_CHAR\_PER\_LINE];  //printf(" 입력행 번호: ");  //scanf("%d", &position);  cin >> position;  //printf(" 입력행 내용: ");  while (getchar() != '\n');  //엔터키 처리  //제대로 입력이 처리되지 않을 때 입력 버퍼를 지워주기 위해 사용하는 함수  fgets(line, MAX\_CHAR\_PER\_LINE, stdin);  //키보드 표준입출력에서 MAX\_CHAR\_PER\_LINE의 길이만큼 line에 문자열을 넣고, 그것을 반환  insert(position, new Node(line));  }  void DeleteLine() { //한 라인 삭제 처리  //printf(" 삭제행 번호: ");  int position;  //scanf("%d", &position);  cin >> position;  delete remove(position);  }  void ReplaceLine() { //한 라인 변경 처리  int position;  char line[MAX\_CHAR\_PER\_LINE];  //printf(" 변경행 번호: ");  //scanf("%d", &position);  cin >> position;  //printf(" 변경행 내용: ");  while (getchar() != '\n');  //엔터키 처리  //제대로 입력이 처리되지 않을 때 입력 버퍼를 지워주기 위해 사용하는 함수  fgets(line, MAX\_CHAR\_PER\_LINE, stdin);  replace(position, new Node(line));  //fflush(stdin); -> 불필요한 작업  }  void LoadFile(const char\* fname) {  FILE\* fp = fopen(fname, "r"); //fname을 읽기 전용으로 엽니다.  if (fp != NULL) {  char line[MAX\_CHAR\_PER\_LINE];  while (fgets(line, MAX\_CHAR\_PER\_LINE, fp)) {  insert(size(), new Node(line));  }  }  fclose(fp);  }  void StoreFile(const char\* fname) {  FILE\* fp = fopen(fname, "w"); //fname을 쓰기 전용으로 엽니다.  if (fp != NULL) {  Display(fp);  fclose(fp);  }  printf("\n");  }  void Findreplace() {  char finding[MAX\_CHAR\_PER\_LINE]; //찾을 문자열  char line[MAX\_CHAR\_PER\_LINE]; //바꿀 문자열 복사  //char line[MAX\_CHAR\_PER\_LINE]; //바꿀 문자열  //printf(" 찾을 내용: ");  while (getchar() != '\n');  fgets(finding, MAX\_CHAR\_PER\_LINE, stdin);  //printf(" 바꿀 내용: ");  fgets(line, MAX\_CHAR\_PER\_LINE, stdin);  find\_and\_replace\_number\_return2(finding, new Node(line));  }  };  void Usage() {  //printf("[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료, f-찾아 바꾸기=> ");  }  int main() {  char command = NULL; //변수 초기화  LineEditor editor; //입력값을 담는 객체  do {  command = NULL;  if (command != 'r') { //임시방편  Usage();  }  cin >> command;  //command = getchar();, 디버깅 결과, getchar 쓰면 Usage가 두 번 돌음  switch (command) {  case 'd': editor.DeleteLine(); break;  case 'i': editor.InsertLine(); break;  case 'r': editor.ReplaceLine(); break;  case 'l': editor.LoadFile("c:\\Users\\jeong\\Desktop\\Test.txt"); break;  case 's': editor.StoreFile("c:\\Users\\jeong\\Desktop\\Test.txt"); break;  case 'p': editor.Display(); break;  case 'f': editor.Findreplace(); break;  }  //fflush(stdin); //입력 버퍼 비우기  } while (command != 'q');  if (command == 'q') {  int sum = 0;  for (int i = 0; i < stringcount; i++) {  if (editorcopy[i] == "EOF") {  cout << editorcopy[i] << "\n";  sum = 0;  }  else {  cout << sum << ": " << editorcopy[i];  sum++;  }  }  }  return 0;  } |

-난점

1. findReplace 제작의 어려움: findReplace는 값만 보고 리스트의 data를 찾아야 함

2. while문 사용의 직관적인 이해 어려움: 파일 포인터 이해와 관련, fflush가 안 되는 것

3. p와 q는 명령 자체가 직관적으로 이해되지 않음(명령 p는 왜 저장하는 것인데 출력인가? 명령 q는 왜 종료인데 출력인가?)

4. 문자열 외에는 q 명령 구현이 어려움: EOF 입출력의 문제, 상속받은 객체들의 복사가 어려움, 출력 시 “\n”까지 신경써야 함. 그리고 무엇보다 번호를 매기는 변수를 따로 두어야 하고, 그 조건도 까다로움

5. 전반적으로 코드 길이가 너무 길어서, 수정하기 매우 어려움: STL이 중요한 이유…연결 리스트에도 STL 있어요…

->리스트보다 연결 리스트 문제, 차라리 연결 리스트로 구현한 스택, 큐, 덱이 더 연계성이 있고 난이도가 더 수용 가능하지 않았을까?