|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования FPMI_ngtu_neti_rgb_polya«Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Практическое задание № 1 | | |
| по дисциплине «Структуры данных и алгоритмы» | | |
| **Линейные списки** | | |
|  | | |
|  | Бригада 4 | Шахрай вадим |
| Группа ПМ-15 | Семенова Варвара |
| Вариант 4 | Рассадин святослав |
|  |  |
|  |  |
|  |  |
| Преподаватели | Тракимус Юрий Викторович |
|  |  |
| Новосибирск, 2022 | | |

1. **Условие задачи**

Для заданных полиномов Pn(x) и Qn(x) найти полином R – сумму полиномов P и Q. Каждый полином представить в виде списка:

n

n-1

an-1

1

a1

a1

0

⏚

P

…

an

Причем в список не включать коэффициенты равные нулю. Считать, что входные данные не содержат равных нулю коэффициентов.

1. **Анализ задачи**

Входные данные: polynomials.txt – файл с двумя полиномами.

Выходные данные: Pn(x), Qn(x), Rn(x) полиномы.

Д ля решения задачи определяем структуру **term**, представляющую собой модель одночлена полинома. Структура имеет три поля: степень одночлена n, коэффициент a и указатель на следующий одночлен полинома next. Сначала объявляем указатели P и Q, которые будут указывать на первый одночлен их полинома, и указатель R. Далее с помощью функции **scanPolynomials** считываем полиномы из текстового файла в P и Q.

Принцип работы функции scanPolynomials. Параметры функции: path – путь к файлу, p – указатель на указатель, в который нужно записать первый одночлен первого полинома, q – указатель на указатель, в который нужно записать первый одночлен второго полинома. Открываем поток для чтения файла по пути, указанном в path, с помощью стандартной функции fopen\_s. С потоком работаем только в том случае, если его удалось открыть, т.е. он не равен NULL. Вводим 3 специальные переменные: sign, отвечающий за знак одночлена, end – "флаг", обозначающий, что полином прочитан в файле до конца, и логический p\_end, означающий, что полином p построен. Инициализируем 3 переменные: a – коэффициент только что прочитанного одночлена, N – наибольшая степень одночлена любого полинома, n – степень только что прочитанного одночлена. Создаём указатель r, через который осуществляется доступ к каждому одночлену полинома p или q.

Далее в цикле считываем каждый одночлен полиномов отдельно, пока не дойдем до конца файла. На каждой итерации цикла выполняем следующие действия:

1. Если степень у прочитанного одночлена больше ранее зафиксированной, то фиксируем эту степень как новый максимум.
2. Если перед одночленом стоит знак '-', делаем его коэффициент отрицательным.
3. Если первый одночлен полинома меньше n, то нужно добавить этот одночлен с большей степенью в самое начало полинома. Делаем это через цикл:
   1. Добавляем новые одночлены в полином на первое место, пока не добавим тот, чья степень стала максимальной для этого полинома.
   2. Совершив переход к нужному одночлену, прибавляем его коэффициент. Таким образом, мы можем складывать подобные одночлены, т.е. приводить полином к более простому виду.
4. Если прочитанный одночлен имеет равную или меньшую степень относительно первого одночлена полинома, то перемещаемся по полиному r от начала до одночлена со степенью равной n через указатель t (он нужен, чтобы не сбить ссылку на первый одночлен у r), и прибавляем его коэффициент к нужному одночлену.
5. Если уже дочитали до конца первого полинома, говорим, что первый полином построен, и переназначаем r, чтобы начать заполнять второй полином.
6. После этого обновляем полином p или q, в зависимости от того, заполнен ли уже полином p.

После того, как файл прочитан до конца, а значит, оба полинома заполнены, если степень одного из полиномов (*p1*) больше, чем степень другого (*p2*), то добавляем одночлены в начало полинома *p2*, пока степень *p2* меньше *p1*. Результатом работы функции будут 2 приведённых и отсортированных полинома. Графическая иллюстрация работы сортировки:

18x2

- Одночлен, прочитанный из файла на текущей интерации цикла.

nxk

- Одночлен, добавленный в полином.

- Одночлен, подобный одночлену прочитанному из файла.

- В nxk есть укзатель на mxj.

18x2

0x1

0x0

-21x3

-21x3

18x2

0x1

0x0

nxk

nxk

nxk

mxj

Для примера возьмём: 18x2 - 21x3 + 24x0 - 6x3

- Демонстрация перехода от В nxk (cxh) к zxw (axp).

nxk

mxj

axp

bxt

zxw

cxh

-6x3

-27x3

18x2

0x1

24x0

24x0

-21x3

18x2

0x1

24x0

После приведения и сортировки полиномы готовы к суммированию. Сложение полиномов реализует функция addPolynomials.

Принцип работы функции addPolynomials. Параметры функции: a и b - 2 указателя на полиномы (передается P и Q). Результат работы функции: указатель на первый член полинома, получившегося при суммировании полиномов a и b. Создаем указатель r на первый элемент полинома суммы. С помощью цикла переберём все члены a и b, а также добавим члены в r. Создаём указатель p, указываем ему адрес, на который указывает r. Через p будем добавлять элементы в r на каждой итерации цикла, тем самым сохранив адрес, на который указывает r. В конце итерации мы “сдвигаем” указатели всех 3 полиномов на их следующий элемент. Условием для завершения цикла будет равенство указателя a нулевому адресу (т.е. были перебраны все члены полиномов). Так, мы устанавливаем, что цикл пройдет от N-го члена полиномов до 0-го. В цикле заполняем член полинома r данными из p. Графическое представление работы цикла (**N** – степень N-го члена, aN и bN – коэффициенты N-го члена полиномов a и b соответственно):

⏚

r

an + bn

an-1 + bn-1

an-2 + bn-2

0

N

N-1

N-2

0

an + bn

an-1 + bn-1

0

N

N-1

0

r

⏚

an + bn

0

N

0

r

⏚

1-ая итерация

2-ая итерация

3-ая итерация и т.д.

После того как цикл завершился, возвращаем r.

Последний шаг – вывести получившиеся полиномы на консоль. Эту задачу выполняет функция printPolynomial. В качестве параметров функция принимает указатель на первый член полинома. С помощью цикла перебирается каждый член полинома и выводится на консоль, в форматированном виде.

1. **Текст программы**

#include <conio.h>

#include <stdio.h>

#include <locale.h>

#include <windows.h>

/// <summary>

/// Структура представляет собой модель одночлена.

/// </summary>

struct term {

term(UCHAR \_n = 0, float \_a = 0, term \*\_next = NULL) : n(\_n), a(\_a), next(\_next) { }

/// <summary>

/// Степень.

/// </summary>

UCHAR n;

/// <summary>

/// Коэффициент.

/// </summary>

float a;

/// <summary>

/// Указатель на следующий одночлен полинома.

/// </summary>

term \*next;

};

/// <summary>

/// Функция возвращает знак числа.

/// </summary>

/// <param name="n">Число, у которого нужно определить знак.</param>

/// <returns>'+' или '-'.</returns>

char sign(float n);

/// <summary>

/// Функция возвращает модуль числа.

/// </summary>

/// <param name="n">Число.</param>

/// <returns>Модуль числа.</returns>

float abs(float n);

/// <summary>

/// Выводит полином на консоль.

/// </summary>

/// <param name="p">Указатель на первый одночлен полинома.</param>

/// <param name="c">Имя полинома.</param>

void printPolynomial(term \*p, char c);

/// <summary>

/// Функция складывает 2 полинома друг с другом.

/// </summary>

/// <param name="a">Указатель на первый полином.</param>

/// <param name="b">Указатель на второй полином.</param>

/// <returns>Указатель на полином, который будет результатом сложения первого и второго полиномов.</returns>

term \* addPolynomials(term \*a, term \*b);

/// <summary>

/// Функция считывает из определенного файла 2 полинома.

/// </summary>

/// <param name="path">Путь к файлу.</param>

/// <param name="p">Указатель на указатель, в который нужно записать первый одночлен одного полинома.</param>

/// <param name="q">Указатель на указатель, в который нужно записать первый одночлен другого полинома.</param>

void scanPolynomials(const char \*path, term \*\*p, term \*\*q);

int main()

{

setlocale(0, "");

term \*P = new term(), \*Q = new term(), \*R;

scanPolynomials("D:/polynomials.txt", &P, &Q);

R = addPolynomials(P, Q);

printPolynomial(P, 'P');

printPolynomial(Q, 'Q');

printPolynomial(R, 'R');

return 0 \* \_getch();

}

void scanPolynomials(const char \*path, term \*\*p, term \*\*q)

{

FILE \*stream;

fopen\_s(&stream, path, "r");

if (stream)

{

// sign нужен для определения знака коэффициента, end - "флаг", обозначающий, что полином прочитан в файле до конца, если end = '\n'.

char sign = '+', end = ' ';

float a = 0; // Коэффициент только что прочитанного одночлена.

UCHAR N = 0, n = 0; // N - наибольшая степень одночлена любого полинома, n - степень только что прочитанного одночлена.

bool p\_end = false; // Означает, что полином p построен.

// r - указатель, с помощью которого осуществляется доступ к каждому одночлену полинома p или q. Читаем каждый одночлен полиномов отдельно, пока не дойдем до конца файла (EOF).

for (term \*r = new term(); fscanf\_s(stream, " %c %fx^%hhi%c", &sign, 1, &a, &n, &end, 1) != EOF; )

{

// Если степень у прочитанного одночлена больше ранее зафиксированной, то зафиксировать эту степень, как новый максимум.

if (N < n) N = n;

// Если перед одночленом стоит знак '-', отрицаем коэффициент.

if (sign == '-') a \*= -1;

// Если первый одночлен полинома меньше n, то в полином нужно добавить этот одночлен с большей степенью в самое начало полинома.

if (n > r->n)

{

// Добавляем новые одночлены в полином на первое место, пока не добавим тот, чья степень стала максимальной для этого полинома.

for ( ; r->n != n; ) r = new term(r->n + 1, 0, r);

// Совершен переход к нужному одночлену, прибавляем его коэффициент. \*Такой способ задания коэффициента (+=) для одночлена позволяет складывать подобные одночлены.

r->a += a;

}

else // Иначе, значит, что прочитанный одночлен имеет равную или меньшую степень относительно первого одночлена полинома.

{

term \*t = r; // Указатель нужен, чтобы не сбить ссылку на первый одночлен у r.

for ( ; t->n != n; t = t->next); // Перемещаемся по полиному r от начала до одночлена со степенью равной n.

t->a += a; // Совершен переход к нужному одночлену, прибавляем его коэффициент.

}

// Если уже дочитали до конца первого полинома, говорим, что первый полином построен, и переназначаем r, чтобы заполнять второй полином.

if (end == '\n' && !p\_end)

{

p\_end = true;

r = new term();

}

p\_end ? \*q = r : \*p = r; // Обновляем полином p или q, в зависимости от того, заполнен ли уже полином p.

}

// Если степень одного из полиномов (p1) больше, чем степень другого (p2), то добавляем одночлены в начало полинома p2, пока степень p2 < p1.

for (term \*\*z = (\*p)->n < N ? p : (\*q)->n < N ? q : NULL; z && (\*z)->n < N; )

\*z = new term((\*z)->n + 1, 0, \*z);

fclose(stream);

}

}

term \* addPolynomials(term \*a, term \*b)

{

// Указатель на первый элемент полинома суммы.

term \*r = new term();

// На каждой итерации переходит к следующему элементу каждого полинома, пока не дойдет до последнего, у которого указатель на следующий равен NULL.

for (term \*p = r; a; p = p->next, a = a->next, b = b->next)

{

p->n = a->n;

p->a = a->a + b->a;

p->next = a->next ? new term() : NULL;

}

return r;

}

void printPolynomial(term \*p, char c)

{

bool f = false; // Индикатор "есть одночлен перед текущим".

printf\_s("%cn(x) = ", c);

for ( ; p; p = p->next)

{

if (p->n && p->a) // Если степень != 0 и коэф != 0, то

{

if (abs(p->a) != 1) // если |коэф| != 1, то

// если есть предодночлен, пишем " %с |%а|", иначе пишем "%а".

f ? printf\_s(" %c %g", sign(p->a), abs(p->a)) : printf\_s("%g", p->a);

// иначе, если есть предодночлен пишем " %с ", иначе, "%с".

else printf\_s(f ? " %c " : "%c", sign(p->a));

// если степень больше 1, то пишем "x^%n", иначе - "х".

p->n > 1 ? printf\_s("x^%d", p->n) : printf\_s("x");

f = true;

}

else if (!p->n) // иначе, если степень == 0, то

{

if (p->a && f) // если коэф != 0 и есть предодночлен, то

printf\_s(" %c %g", sign(p->a), abs(p->a));

else if(!f) printf\_s("%g", p->a); // иначе, если нет предодночлена, "%a".

printf\_s(".\n");

}

}

}

char sign(float n)

{

return n > 0 ? '+' : '-';

}

float abs(float n)

{

return sign(n) == '-' ? -1 \* n : n;

}

1. **Набор тестов**

|  |  |  |
| --- | --- | --- |
| **№** | **Входные данные** | **Назначение** |
| 1 | - 45x^6 - 54x^5 - 12x^4 + 16x^3 + 6x^2 - 1x^1 + 3x^0  + 17x^6 - 18x^5 + 4x^4 + 66x^3 + 21x^2 - 73x^1 - 101x^0 | Проверка корректности работы программы. |
| 2 | - 48x^4 - 18x^3 + 1x^2 - 6x^1 - 102x^0  + 2x^4 + 18x^3 + 63x^2 - 1x^1 + 102x^0 | Сумма коэффициентов при x^3 и x^0 равна 0, она не должна выводиться. Проверка работы вывода. |
| 3 | + 47x^0 + 24x^1 - 60x^2 - 1x^3 + 7x^4 + 85x^5 - 16x^6  - 81x^0 - 37x^1 + 76x^2 - 2x^3 + 110x^4 + 8x^5 + 100x^6 | Обратно упорядоченные полиномы. Проверка работы сортировки. |
| 4 | + 49,5x^2 - 30x^1 - 18,9x^4 - 57x^3 - 67x^0 + 84x^5 - 92x^6  + 45x^1 + 91x^6 + 5,14x^2 + 16x^5 + 17,9x^4 + 48x^3 + 68x^0 | Частично упорядоченные полиномы. Проверка работы сортировки. |
| 5 | - 54x^6 - 63x^5 - 13x^4 + 66x^3 - 7x^2 - 111x^1 + 8x^0  + 54x^6 + 63x^5 + 13x^4 - 66x^3 + 7x^2 + 111x^1 - 8x^0 | Сумма всех коэффициентов равна 0. Проверка работы вывода. |

1. **Результаты работы программы**

|  |  |
| --- | --- |
| **№** | **Ввод/вывод программы** |
| 1 | Pn(x) = -45x^6 - 54x^5 - 12x^4 + 16x^3 + 6x^2 - x + 3.  Qn(x) = 17x^6 - 18x^5 + 4x^4 + 66x^3 + 21x^2 - 73x - 101.  Rn(x) = -28x^6 - 72x^5 - 8x^4 + 82x^3 + 27x^2 - 74x - 98. |
| 2 | Pn(x) = -48x^4 - 18x^3 + x^2 - 6x - 102.  Qn(x) = 2x^4 + 18x^3 + 63x^2 - x + 102.  Rn(x) = -46x^4 + 64x^2 - 7x. |
| 3 | Pn(x) = -16x^6 + 85x^5 + 7x^4 - x^3 - 60x^2 + 24x + 47.  Qn(x) = 100x^6 + 8x^5 + 110x^4 - 2x^3 + 76x^2 - 37x - 81.  Rn(x) = 84x^6 + 93x^5 + 117x^4 - 3x^3 + 16x^2 - 13x - 34. |
| 4 | Pn(x) = -92x^6 + 84x^5 - 18,9x^4 - 57x^3 + 49,5x^2 - 30x - 67.  Qn(x) = 91x^6 + 16x^5 + 17,9x^4 + 48x^3 + 5,14x^2 + 45x + 68.  Rn(x) = -x^6 + 100x^5 - x^4 - 9x^3 + 54,64x^2 + 15x + 1. |
| 5 | Pn(x) = -54x^6 - 63x^5 - 13x^4 + 66x^3 - 7x^2 - 111x + 8.  Qn(x) = 54x^6 + 63x^5 + 13x^4 - 66x^3 + 7x^2 + 111x - 8.  Rn(x) = 0. |