ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»**

Московский институт электроники и математики им. А.Н. Тихонова

Департамент прикладной математики

Отчёт
по лабораторной работе №10
по курсу «Алгоритмизация и программирование»
Задание № 13

ФИО студента	Номер группы	Дата
Кейер Александр Петрович	БПМ-231	6 июня 2024 г.

Москва, 2024

1 Задание (вариант № 13)

Номер варианта	**Постановка задачи:** Многочлен с целыми коэффициентами можно представить в виде списка. Задать многочлен от X **односвязным списком**. Элемент списка содержит показатель степени X и ненулевой коэффициент при этой степени (в списке не должно быть элементов с одинаковыми степенями). Составить программу, включающую помимо указанных в задании функций, функции создания и вывода списка на экран. Список или списки должны отображаться на экране до обработки и после.	**Постановка задачи:** Вводится строка из строчных латинских букв, слова разделены пробелами, признак конца ввода - точка. При выполнении задания предложение организовать в виде **двухсвязного списка** слов. Программа должна содержать функции формирования исходного списка, вывода списка до и после модификации и реализации непосредственно варианта задания.
	L=/-/x^-x^-x-x^	
13	Написать функцию, формирующую многочлен (список) из двух многочленов L1, L2 по следующему правилу: в новый многочлен включаются те элементы, степени которых входят в оба многочлена, при этом выбирается наибольший из коэффициентов. Например, $L1=2x^2-4x^5+7$ и $L2= -5x^2+x^3+x-3x^5$, тогда $L= 2x^2-3x^5$	Перед минимальным словом (в лексикографическом смысле) вставить элемент с его инверсией.

2 10.1

```c
#include <stdio.h> // Input/output library.
#include <stdlib.h> // Memory allocation.
#include <assert.h> // Assertion library.
#include <string.h> // String functions library.

struct M {
  long a;
  unsigned n;
};
typedef struct M M;

struct PListItem {
  struct M* valueP;
  struct PListItem* nextP;
};
typedef struct PListItem PListItem;

struct PList {
  PListItem* headP;
};
typedef struct PList PList;

long parseStringToLongInt(char* s) {
  long out = 0;
  int placeholder10 = 1;
  int sLength = strlen(s);
  int k = 1;

  if (s[0] == '-') {
    k = -1;
  }

  for (int i = sLength - 1; i >= 0; i--) {
    if (s[i] == ' ' || s[i] < '0' || s[i] > '9') {
      continue;
    }

    out += k * placeholder10 * (s[i] - '0');
    placeholder10 *= 10;
  }

  return out;
}
```

```
44
45    void printPList(PList* pListP) {
46      PListItem* cur = pListP->headP;
47
48      if (cur == NULL) {
49        printf("unfortunately this polynomial list is
    incorrect or empty =(\n");
50      }
51
52      char tmp = '+';
53
54      while (cur != NULL) {
55        if (cur->valueP->a < 0) {
56          tmp = 0;
57        } else {
58          tmp = '+';
59        }
60
61        printf("%c%ldx^%u", tmp, cur->valueP->a, cur->valueP
    ->n);
62
63        cur = cur->nextP;
64      }
65
66      printf("\n");
67    }
68
69    PListItem* findWithSameN(PList* pList, int n) {
70      PListItem* cur = pList->headP;
71
72      while (cur != NULL) {
73        if (cur->valueP->n == n) {
74          return cur;
75        }
76
77        cur = cur->nextP;
78      }
79
80      return NULL;
81    }
82
83    PListItem* clearPListItem(PListItem* pListItem) {
84      PListItem* nextP = pListItem->nextP;
85
86      free(pListItem->valueP);
```

```
87        free(pListItem);
88
89        return nextP;
90      }
91
92      PList* clearPList(PList* pList) {
93        PListItem* cur = pList->headP;
94
95        while (cur != NULL) {
96          cur = clearPListItem(cur);
97        }
98
99        free(pList);
100       pList = NULL;
101
102       return pList;
103     }
104
105     void clearMonomWithA0(PList* pList) {
106       PListItem* cur = pList->headP->nextP;
107       PListItem* prev = pList->headP;
108
109       while (cur != NULL) {
110         if (cur->valueP->a == 0) {
111           prev->nextP = cur->nextP;
112           clearPListItem(cur);
113         } else {
114           prev = prev->nextP;
115         }
116
117         cur = prev->nextP;
118       }
119
120       if (pList->headP->valueP->a == 0) {
121         cur = pList->headP;
122         pList->headP = pList->headP->nextP;
123         clearPListItem(cur);
124       }
125     }
126
127     PList* parsePStringToPList(char* pStringP) {
128       PList* out = (PList*)malloc(sizeof(PList));
129       out->headP = (PListItem*)malloc(sizeof(PListItem));
130       out->headP = NULL;
131
```

```
132       if (pStringP[0] == 0) {
133         return out;
134       }
135 
136       PListItem* sameNCandidate = NULL;
137       PListItem* prev = NULL;
138       PListItem* cur = (PListItem*)malloc(sizeof(PListItem));
139       cur->valueP = (M*)malloc(sizeof(M));
140       cur->nextP = NULL;
141 
142       char placeholderP[100] = "";
143 
144       for (int i = 0; pStringP[i] != 0; i++) {
145         if (pStringP[i] == ' ') {
146           continue;
147         }
148 
149         if ((pStringP[i] == '+' || pStringP[i] == '-')) {
150           if (placeholderP[0] != 0) {
151             cur->valueP->n = parseStringToLongInt(
    placeholderP);
152 
153             PListItem* sameNCandidate = findWithSameN(out,
    cur->valueP->n);
154 
155             if (sameNCandidate == NULL) {
156               if (out->headP == NULL) {
157                 out->headP = cur;
158               } else {
159                 prev->nextP = cur;
160               }
161 
162               prev = cur;
163             } else {
164               sameNCandidate->valueP->a += cur->valueP->a;
165               clearPListItem(cur);
166             }
167 
168             cur = (PListItem*)malloc(sizeof(PListItem));
169             cur->valueP = (M*)malloc(sizeof(M));
170             cur->nextP = NULL;
171           }
172 
173           placeholderP[0] = pStringP[i];
174           placeholderP[1] = 0;
```

```
175         }
176
177         if (pStringP[i] == 'x') {
178           cur->valueP->a = parseStringToLongInt(placeholderP)
    ;
179           assert(cur->valueP->a != 0);
180
181           placeholderP[0] = 0;
182         }
183
184         if (pStringP[i] <= '9' && pStringP[i] >= '0') {
185           placeholderP[strlen(placeholderP) + 1] = 0;
186           placeholderP[strlen(placeholderP)] = pStringP[i];
187         } else if (
188         pStringP[i] != 'x'
189         && pStringP[i] != '+'
190         && pStringP[i] != '-'
191         && pStringP[i] != '^'
192         ) {
193           out->headP = NULL;
194           return out;
195         }
196       }
197
198       cur->valueP->n = parseStringToLongInt(placeholderP);
199
200       sameNCandidate = findWithSameN(out, cur->valueP->n);
201
202       if (sameNCandidate == NULL) {
203         if (out->headP == NULL) {
204           out->headP = cur;
205         } else {
206           prev->nextP = cur;
207         }
208       } else {
209         sameNCandidate->valueP->a += cur->valueP->a;
210         clearPListItem(cur);
211       }
212
213       clearMonomWithA0(out);
214
215       return out;
216     }
217
218     void copyPListItem(PListItem* new, PListItem* cur) {
```

```
219       new->nextP = NULL;
220       new->valueP->a = cur->valueP->a;
221       new->valueP->n = cur->valueP->n;
222     }
223
224     PList* mixTwoPLists(PList* pListP1, PList* pListP2) {
225       PList* out = (PList*)malloc(sizeof(PList));
226
227       PListItem* cur1 = pListP1->headP;
228       PListItem* cur2 = pListP2->headP;
229
230       PListItem* prevGood = NULL;
231
232       out->headP = NULL;
233
234       while (cur1 != NULL) {
235         cur2 = pListP2->headP;
236
237         while (cur2 != NULL && cur2->valueP->n != cur1->
    valueP->n) {
238           cur2 = cur2->nextP;
239         }
240
241         if (cur2 != NULL) {
242           PListItem* curGood = (PListItem*)malloc(sizeof(
    PListItem));
243           curGood->valueP = (M*)malloc(sizeof(M));
244
245           if (cur2->valueP->a >= cur1->valueP->a) {
246             copyPListItem(curGood, cur2);
247           } else {
248             copyPListItem(curGood, cur1);
249           }
250
251           if (out->headP == NULL) {
252             out->headP = curGood;
253           } else {
254             prevGood->nextP = curGood;
255           }
256
257           prevGood = curGood;
258         }
259
260         cur1 = cur1->nextP;
261       }
```

```
262
263       return out;
264     };
265
266     void readPStringFromUser(int pi, char* pStringP) {
267       printf("Enter P%d: ", pi);
268
269       fflush(stdin);
270       fgets(pStringP, 100, stdin);
271       pStringP[strcspn(pStringP, "\n")] = 0;
272     }
273
274     int main() {
275       char* pStringP1 = (char*)malloc(100 * sizeof(char));
276       char* pStringP2 = (char*)malloc(100 * sizeof(char));
277
278       printf("Lab 10. Keyer, BAM231.\n");
279
280       readPStringFromUser(1, pStringP1);
281       readPStringFromUser(2, pStringP2);
282
283       PList* pListP1 = parsePStringToPList(pStringP1);
284       PList* pListP2 = parsePStringToPList(pStringP2);
285
286       printPList(pListP1);
287       printPList(pListP2);
288
289       PList* finalPList = mixTwoPLists(pListP1, pListP2);
290
291       printf("Polynomials mixin: ");
292       printPList(finalPList);
293
294
295       clearPList(pListP1);
296       clearPList(pListP2);
297       clearPList(finalPList);
298
299       finalPList->headP = NULL;
300
301       free(pStringP1);
302       free(pStringP2);
303
304       return 0;
305     }
306
```

3 10.1

```c
#include <stdio.h> // Input/output library.
#include <stdlib.h> // Memory allocation.
#include <string.h> // String functions library.

struct WordListItem {
  char* valueP;
  struct WordListItem* nextP;
  struct WordListItem* prevP;
};
typedef struct WordListItem WordListItem;

struct WordList {
  WordListItem* headP;
  WordListItem* tailP;
};
typedef struct WordList WordList;

// Function validating a string.
int isValid(char* s) {
  for (int i = 0; s[i] != 0; i++) {
    if ((s[i] < 'a' || s[i] > 'z') && s[i] != ' ' && s[i] != '.') {
      printf("String contains incorrect symbol: %c\n", s[i]);
      return 0;
    }
  }

  return 1;
}

WordList* parseStringToWordList(char* s) {
  WordList* out = (WordList*)malloc(sizeof(WordList));

  WordListItem* cur = (WordListItem*)malloc(sizeof(WordListItem));
  out->headP = cur;
  WordListItem* prev = NULL;

  cur->valueP = (char*)malloc(sizeof(char) * 100);
  cur->valueP[0] = 0;
  cur->prevP = NULL;

```

```
41      for (int i = 0; s[i] != 0; i++) {
42        if (s[i] != ' ') {
43          cur->valueP[strlen(cur->valueP) + 1] = 0;
44          cur->valueP[strlen(cur->valueP)] = s[i];
45        } else {
46          prev = cur;
47
48          cur->nextP = (WordListItem*)malloc(sizeof(
    WordListItem));
49          cur = cur->nextP;
50
51          cur->prevP = prev;
52          cur->valueP = (char*)malloc(sizeof(char) * 100);
53          cur->valueP[0] = 0;
54        }
55      }
56
57      cur->prevP = prev;
58      cur->nextP = NULL;
59
60      out->tailP = cur;
61
62      return out;
63    };
64
65    void printWordList(WordList* wordList) {
66      if (wordList->headP == NULL) {
67        printf("Incorrect word list was input");
68        return;
69      }
70
71      WordListItem* cur = wordList->headP;
72
73      printf("Next direction: ");
74
75      printf("%s", cur->valueP);
76      cur = cur->nextP;
77
78      while (cur != NULL) {
79        printf(" %s", cur->valueP);
80        cur = cur->nextP;
81      }
82
83      printf(".\n");
84
```

```
85        printf("Prev direction: ");
86
87        cur = wordList->tailP;
88
89        printf("%s", cur->valueP);
90        cur = cur->prevP;
91
92        while (cur != NULL) {
93          printf(" %s", cur->valueP);
94          cur = cur->prevP;
95        }
96
97        printf(".\n");
98      };
99
100     char* inverseWord(char* word) {
101       char* out = (char*)malloc(100 * sizeof(char));
102
103       int wordLength = strlen(word);
104
105       out[wordLength] = 0;
106
107       for (int i = 0; word[i] != 0; i++) {
108         out[wordLength - i - 1] = word[i];
109       }
110
111       return out;
112     }
113
114     WordListItem* findMinimumWordItem(WordList* wordList) {
115       WordListItem* out = wordList->headP;
116       WordListItem* good = out;
117       WordListItem* cur = good;
118
119       while (cur != NULL) {
120         if (strcmp(cur->valueP, good->valueP) <= 0) {
121           good = cur;
122         }
123         cur = cur->nextP;
124       }
125
126       return good;
127     }
128
129     void solution(char* s) {
```

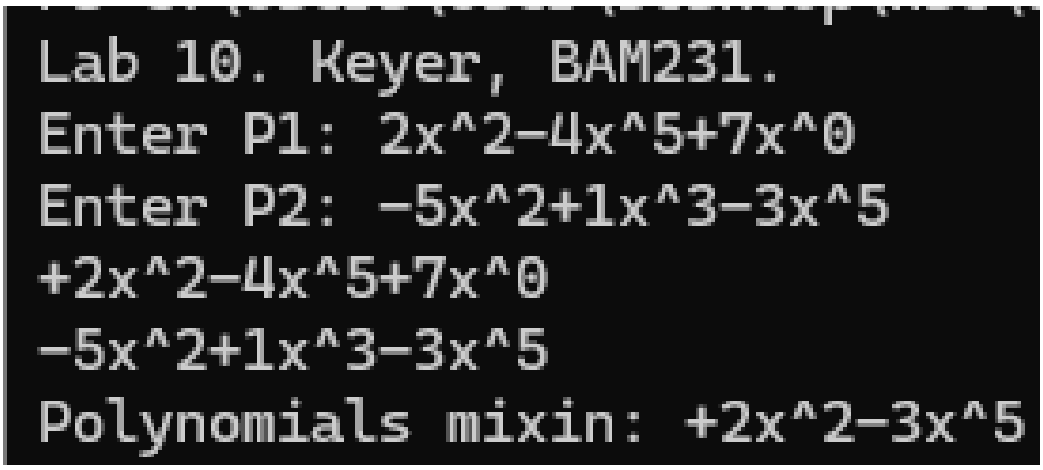
```
130       WordList* wordList = parseStringToWordList(s);
131
132       WordListItem* minimumWordItem = findMinimumWordItem(
    wordList);
133       WordListItem* insertItem = (WordListItem*)malloc(sizeof
    (WordListItem));
134
135       insertItem->valueP = inverseWord(minimumWordItem->
    valueP);
136
137       insertItem->prevP = minimumWordItem->prevP;
138       insertItem->nextP = minimumWordItem;
139
140       if (minimumWordItem->prevP) {
141         minimumWordItem->prevP->nextP = insertItem;
142       } else {
143         wordList->headP = insertItem;
144       }
145
146       minimumWordItem->prevP = insertItem;
147
148       printf("Modifed string:\n");
149       printWordList(wordList);
150     }
151
152     // Function reading a string from stdin.
153     char* readingStringFromUser() {
154       printf("Please, enter correct string: ");
155
156       char* s = (char*)malloc(sizeof(char) * 100);
157
158       fflush(stdin);
159       fgets(s, 100, stdin);
160       s[strcspn(s, "\n")] = 0;
161       s[strcspn(s, ".")] = 0;
162
163       return s;
164     };
165
166     int main() {
167       char* s = readingStringFromUser();
168
169       if (!isValid(s)) {
170         return 1;
171       }
```

```
172
173       printf("Work with string: %s.\n", s);
174       solution(s);
175
176       free(s);
177       return 0;
178     };
179
```

4 Тесты

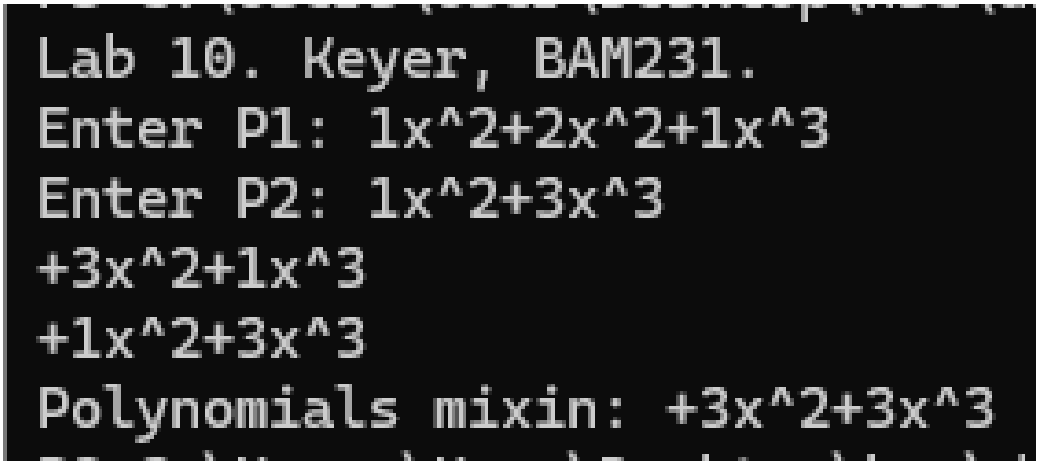
5 10.1

5.1 Тест 1



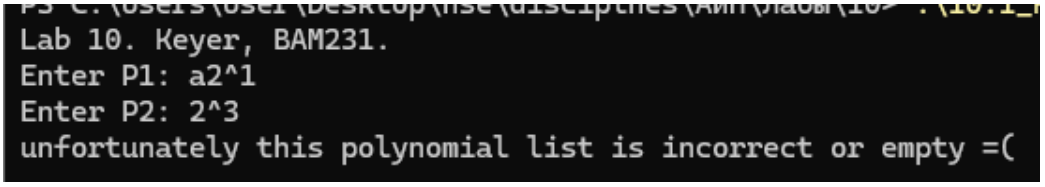
С базовым тестом все нормик и работает! Проверим корнер кейсы.

5.2 Тест 2



Тут, как и ожидалось, слагаемые в P1 с одинаковыми степенями сложились.

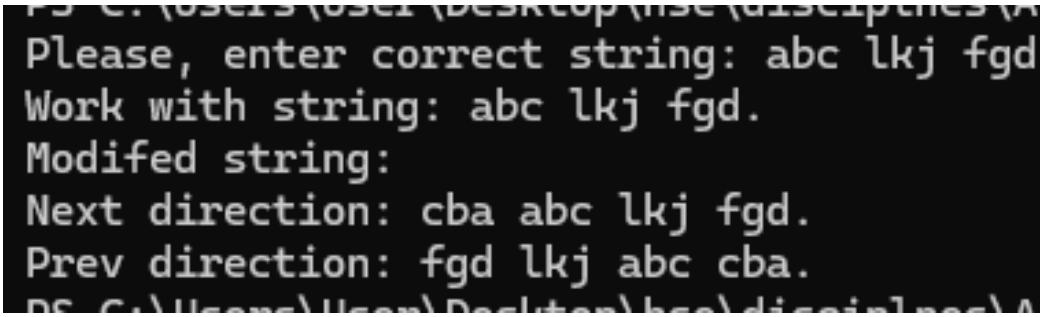
5.3 Тест 3



Ругаемся, если введены неправильные символы или что-то противоречащее условиям задания.

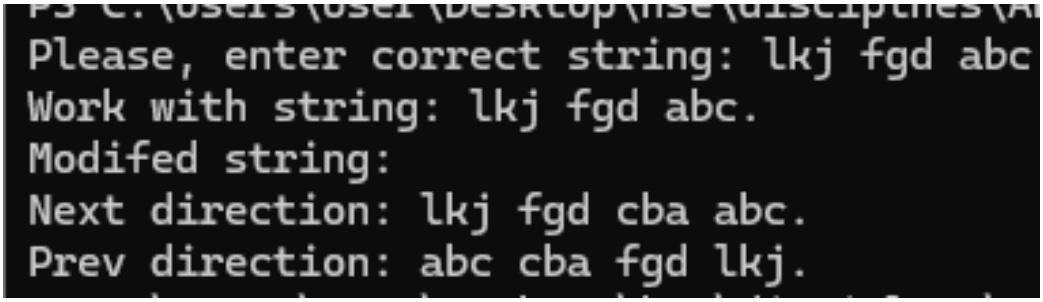
6 10.2

6.1 Тест 1



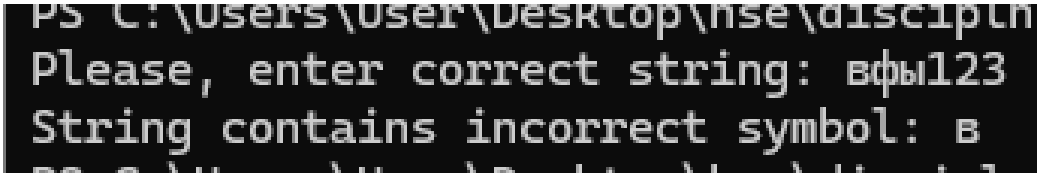
С базовым тестом все нормик и работает! Проверим корнер кейсы.

6.2 Тест 2



Тут все, как и ожидалось.

6.3 Тест 3



Ругаемся, если введены неправильные символы или что-то противоречащее условиям задания.