МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студентка гр. 7383 _____ Рудоман В.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

СОДЕРЖАНИЕ

Цель работы

Изучить иерархические списки. Научиться использовать иерархические списки в программах.

Задание

Пусть выражение (логическое, арифметическое, алгебраическое*) представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в префиксной форме ((<операция> <аргументы>)), либо в постфиксной форме (<аргументы> <операция>)).

* - здесь примем такую терминологию: в арифметическое выражение входят операции +, -, *, /, а в алгебраическое – +, -, * и дополнительно некоторые функции.

В заданиях 22 – 24 функции sqrt(), log(,), sin(), cos(), power(,) используются в классической форме (аргумент(ы) в скобках, а слева от скобок – название функции), а не в префиксной или постфиксной!

Вариант 22:

алгебраическое (+, -, *, sqrt(), log(,)), проверка синтаксической корректности, простая проверка log(,), префиксная форма

Постановка задачи

Нужно написать программу, которая считает выражение, запишет его в иерархический список, затем проверит его синтаксическую корректность и выведет сообщение об ошибках или их отсутствии.

Описание алгоритма

1. Функция check_mistakes принимает элемент списка

2. Проверяет его на корректность по ряду критериев. Пишет в файл log.txt, если ошибки были найдены.

3. Переходит на элемент ниже данного, если у данного элемента указатель на нижний элемент не пуст (п.1). Если пуст — пункт 4.

4.Переходит на следующий элемент на уровне, если у данного элемента указатель на следующий не пуст (п.1). Если пуст алгоритм завершается.

Описание функций

1. Element *createNewEl(char* atom, int tag, Element* prev, Element* up)

Функция принимает на вход:

atom – строка, содержащая операцию или аргумент;

tag – целое число, содержащее 0, если элемент список и 1, если атом;

prev – указатель на предыдущий элемент в списке, или на NULL, если такого нет.

up – указатель на верхний элемент в списке или на NULL, если такого нет.

Функция возвращает указатель на созданный элемент списка.

2. Element *makeList(int * level, Element ** up, FILE ** text);

Функция принимает на вход:

level – указатель на целое число, которое содержит значение глубины рекурсии;

up – указатель на верхний элемент, откуда произошло углубление списка:

text – указатель на текстовый файл, где содержится обрабатываемое выражение.

Функция возвращает указатель на элемент списка, конкретно на голову уровня.

3. void clearList(Element * El);

Функция принимает на вход:

El – указатель на элемент списка, конкретно на голову списка, который требуется очистить.

Функция ничего не возвращает.

4. void check_mistakes(Element * El);

Функция принимает на вход:

El – указатель на элемент списка, конкретно на голову списка, который требуется проверить на синтаксическую корректность.

Функция ничего не возвращает, т.к. всю информацию об ошибках записывает в текстовый файл log.txt.

Тестирование

Было сделано 10 тестов для демонстрации и проверки работы программы.

Тест №	Данные	Результат
1	(+ 76 (- 88) log(6,77) (* 87 (+ 88 sqrt(77))))	Syntactically correct!
2	(* sqrt(25) log(26,88) (* 76 (+ 87 (-7))))	Syntactically correct!
3	(+ 34 log(2,87) sqrt(77) (- 9) (-7) (* 4343 8 (+ 66 5)))	Syntactically correct!
4	(+ 76 (- 88 (* 6 log(8,6) (+ 87 66)) sqrt(16))(+ 77 5))	Syntactically correct!
5	(- 99 (+ 88 (* 873 44)) sqrt(87) 87 55 (-8) log(3,87) (* 4 4) (* 45 44 (+ 343 sqrt(66))))	Syntactically correct!
6	(+ log(1,87 7)	Syntactically incorrect! All errors: Brackets do not match! Mistake! Operation + requires >1 arguments! Function log(a,b) requires a != 1! Too many arguments in function log(,)!
7	(* 76 sqrt(76 99)) 99 77	Syntactically incorrect! All errors: Arguments out of brackets! Function sqrt() requires 1 argument!
8	(* sqr(76) (log 66,5)	Syntactically incorrect! All errors: '(' required after function! Brackets do not match!
9	((+ 7) (* 6) (- 77))	Syntactically incorrect! All errors: Mistake! Operation + requires >1 arguments! Mistake! Operation * requires >1 arguments!
10	(+ 76 (-9) + 76 55)	Syntactically incorrect! All errors: Argument (+) is not number! Prefix form of expression required!

Вывод

В ходе лабораторной работы были изучены принципы заполнения иерархических списков и работы с ними. Разработан алгоритм проверки синтаксической корректности алгебраического выражения с префиксной формой записи.

ПРИДОЖЕНИЕ А
ИСХОДНЫЙ КОД

```
#include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 #include <ctype.h>
 #include "functions.h"

 Element * createNewEl(char* atom, int tag, Element* prev, Element* up){

      Element * temp = (Element*)malloc(sizeof(Element));
      if (prev != NULL)
           prev->next = temp;
      if (up != NULL)
           up->down = temp;
      if (tag){
           temp->atom=(char*)malloc(sizeof(char)*5);
           strcpy(temp->atom,atom);
           temp->tag=1;
      }
      else{
           temp->tag=0;
           temp->atom=NULL;
      }
      temp->next=NULL;
      temp->down=NULL;

 return temp;
 }


 Element* makeList(int * level, Element ** up, FILE ** text){
      FILE * log;
      Element * Head;
      Element * El;
      Element * prev = NULL;
      Element * currUp = *up;
      fpos_t txt_pointer;
      int tag;
      int position = 0;
      char symbol;
      char * str_symbol = (char*)malloc(sizeof(char)*5);
      printf("\n");
      for (int i = 0; i < (*level); i++)
```

```
            printf("  ");
      printf("makeList level %d: start\n",(*level)+1);
      for (int i = 0; i < (*level); i++)
            printf("  ");
      printf("Atoms: ");
      while (1){
            tag = 1;
            str_symbol[0] = 0;
            if (position) currUp=NULL;
            fgetpos(*text,&txt_pointer);
            symbol=fgetc(*text);
                  if (*level==0 && position==0 && symbol=='(')
                        symbol=fgetc(*text);
            if (symbol=='\n'){
                  fsetpos(*text,&txt_pointer);
                  return Head;
            }
            //recording of atom (+,-,*,nil,log,sqrt,4536)
            if (isdigit(symbol)){
                  for (int i = 0; i < 5; i++){
                        if (!isdigit(symbol)) break;
                        strncat(str_symbol,&symbol,1);
                        fgetpos(*text,&txt_pointer);
                        symbol = fgetc(*text);
                  }
                  fsetpos(*text,&txt_pointer);
                  symbol = ' ';
            }
            if (isalpha(symbol)){
                  if (position){
                        fsetpos(*text,&txt_pointer);
                        symbol='(';
                        position++;
                  }
                  else{
                        for (int i = 0; i < 5; i++){
                              if (!isalpha(symbol)) break;
                              strncat(str_symbol,&symbol,1);
                              fgetpos(*text,&txt_pointer);
                              symbol = fgetc(*text);
                        }
                        if (symbol != '(' || (strncmp(str_symbol,"log",3)
&& strncmp(str_symbol,"sqrt",4))){
                              log = fopen("log.txt","w+");
```

```
                        if (symbol != '(')
                              fprintf(log,"'(' required after
function!\n");
                        if (strncmp(str_symbol,"log",3) &&
strncmp(str_symbol,"sqrt",4))
                              fprintf(log,"%s is not supported
function!\n",str_symbol);
                        fclose(log);
                  }
                  if (symbol!='('){
                        fsetpos(*text,&txt_pointer);
                  }
                  else symbol=' ';
            }
      }
      switch (symbol){
            case '(':
                  tag=0;
                  strncat(str_symbol,&symbol,1);
                  break;
            case '+':
            case '-':
            case '*':
            case ',':
                  strncat(str_symbol,&symbol,1);
                  break;
            default:
                  break;
      }
//addition of new element
if (str_symbol[0] != 0){
      if (!position){
            Head = createNewEl(str_symbol,tag,prev,currUp);
            prev = Head;
            position++;
            (*level)++;
      }
      else{
            El = createNewEl(str_symbol,tag,prev,currUp);
            prev = El;
            position++;
      }
      if (tag)
            printf("%s -> ",str_symbol);
```

```
     }
     if (symbol == '('){
          printf("nil -> ");
          if (position == 1)
               makeList(level,&Head,text);
          else
               makeList(level,&El,text);
     }
     if (symbol == ')'){
          printf("\n");
          for (int i = 0; i < (*level) - 1; i++)
               printf("  ");
          printf("makeList level %d: end\n",(*level));
          (*level)--;
          return Head;
     }
 }

 return Head;
 }

 void check_mistakes(Element * El){
     int position = 0;
     FILE * log;
     while(El != NULL){
          log = fopen("log.txt","a");
          if (log == NULL){
               log = fopen("log2.txt","w+");
          }
          if (position==0 && El->tag){
               if (El->atom[0]=='+' || El->atom[0]=='*' || El-
>atom[0]=='-'){
                    if (El->next->next == NULL || El->next == NULL)
                         fprintf(log,"Mistake! Operation %c requires >1
arguments!\n",El->atom[0]);
               }

               if (!strncmp(El->atom,"log",3)){
                         if (El->next != NULL){
                              if (!isdigit(El->next->atom[0]))
                                   fprintf(log,"Function log(,)
requires numbers as arguments!\n");
                              if (atoi(El->next->atom) == 1){
```

```
                                        fprintf(log,"Function log(a,b)
requires a != 1!\n");
                                   }
                                   if (atoi(El->next->atom) < 0)
                                        fprintf(log,"Function log(a,b)
requires a > 0!\n");
                                   if (El->next->next != NULL){
                                        if (El->next->next->atom[0] != ',')
                                             fprintf(log,"No ',' in
function log(,)!\n");
                                        if (El->next->next->next != NULL){
                                             if (!isdigit(El->next->next-
>next->atom[0]))
                                                  fprintf(log,"Function
log(,) requires numbers as arguments!\n");
                                             if (atoi(El->next->next->next-
>atom) < 0)
                                                  fprintf(log,"Function
log(a,b) requires b > 0!\n");
                                             if (El->next->next->next->next
!= NULL)
                                                  fprintf(log,"Too many
arguments in function log(,)!\n");
                                        }
                                        else fprintf(log,"Function log(,)
requires 2 arguments, separated with ','!\n");
                                   }
                                   else fprintf(log,"Function log(,)
requires >1 arguments!\n");
                              }
                              else fprintf(log,"Function log(,) requires
arguments!\n");
                   }
                   if (!strncmp(El->atom,"sqrt",4)){
                        if (El->next != NULL){
                             if (!isdigit(El->next->atom[0]))
                                  fprintf(log,"Function sqrt() requires
number as argument!\n");
                             if (El->next->next != NULL)
                                  fprintf(log,"Function sqrt() requires 1
argument!\n");
                        }
                        else fprintf(log,"Function sqrt() requires 1
argument!\n");
```

```
                }
                if (El->atom[0]!='+' && El->atom[0]!='-' && El-
>atom[0]!='*' && El->atom[0]!='s' && El->atom[0]!='l'){
                     if(!isalpha(El->atom[0]))
                          fprintf(log,"Prefix form of expression
required!\n");
                }
           }
           if (position && El->tag){
                if (!isdigit(El->atom[0]) && El->atom[0] != ',')
                     fprintf(log,"Argument (%s) is not number!\n",El-
>atom);
                if (El->atom[0]=='+' || El->atom[0]=='-' || El->atom[0]
== '*')
                     fprintf(log,"Prefix form of expression
required!\n");
           }
           position++;
           fclose(log);
           if (El->down != NULL){
                check_mistakes(El->down);
           }
           El=El->next;
     }
 }

 void clearList(Element * El){
     Element * temp;
     temp=El;
     while (temp != NULL){

           if (temp->down != NULL){
                clearList(temp->down);
           }
           temp=temp->next;
     }
     free(temp);
}
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "functions.h"

#define N 500

int main(){
char * name;
int answer;
FILE * file;

printf("Please input your expression\n");
     scanf("\n");
     name=(char*)malloc(sizeof(char)*N);
     fgets(name,N,stdin);
     file=fopen("input.txt","w+");
     fputs(name,file);
     fclose(file);
     file=fopen("input.txt","r");

fpos_t txt_pointer;
fgetpos(file,&txt_pointer);
fgets(name,N,file);
printf("\nINPUT EXPRESSION:\n");
printf("%s\n",name);
fsetpos(file,&txt_pointer);

printf("\nSTEPS:\n");
char symbol;
int level=0;
Element * Head=NULL;
Element * El;
Element * Up=NULL;
int tag;
Head=makeList(&level,&Up,&file);
symbol=fgetc(file);
fclose(file);

file=fopen("log.txt","a");
if (file == NULL)
     file=fopen("log.txt","w+");
if (level!=0)
     fprintf(file,"Brackets do not match!\n");
```

```
 if (symbol!='\n')
     fprintf(file,"Arguments out of brackets!\n");
 if (Head != NULL) El=Head;
 else{
     printf("List is empty!\n");
     return 0;
 }
 fclose(file);

 name[0]=0;
 check_mistakes(Head);
 file=fopen("log.txt","r");
 if (file==NULL)
     printf("\nSyntactically correct!\n");
 else{
     fgetpos(file,&txt_pointer);
     fgets(name,N,file);
     if (name[0] == 0)
          printf("\nSyntactically correct!\n");
     else{
          fsetpos(file,&txt_pointer);
          printf("\n\nSyntactically incorrect!\nAll errors:\n");
          while (1){
               fgets(name,500,file);
               if (feof(file)) break;
               printf("%s",name);
          }
     }
 }
 free(name);
 fclose(file);
 remove("log.txt");
 remove("input.txt");
 clearList(Head);
 return 0;
}
```

ПРИЛОЖЕНИЕ В
СОДЕРЖАНИЕ ЗАГОЛОВОЧНОГО ФАЙЛА

```
typedef struct Element{
      char* atom;
      int tag;
      struct Element* next;
      struct Element* down;
} Element;

Element * createNewEl(char* atom, int tag, Element* prev, Element* up);

Element* makeList(int * level, Element ** up, FILE ** text);

void clearList(Element * El);

void check_mistakes(Element * El);
```