**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: «Упрощение префиксного выражения и проверка деления на 0 с использованием иерархических списков»**

Студент гр. 7382 Лящевская А. П.

Преподаватель Фирсов М. А.

Санкт-Петербург

2018

**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>) ). Аргументов может быть 1, 2 и более. Например (в префиксной форме): (+ a (\* b (- c))) или (OR a (AND b (NOT c))).

19) Арифметическое выражение, упрощение и проверка деления на 0 (простая), префиксная форма выражения.

**Пояснение задания.**

На вход программе подаётся последовательность чисел и знаков арифметических операций (арифметическое выражение) в форме иерархического списка в префиксной форме. Итогом завершения программы должно быть преобразованное выражение по принципу упрощения, где под принципом упрощения подразумевается сокращение числа лишних операций.

**Описание алгоритма.**

Введённое пользователем выражение прежде всего подлежит проверке на пустоту. И поскольку заданием никакие другие синтаксические проверки не обусловлены, алгоритм продолжает работу, но уже над тем, что хранит выражение.

Затем программа анализирует его структуру, компактно записывает ее в иерархический список и выводит на экран результат в соответствии с ограничивающими условиями сокращения. При обнаружении в выражении или подвыражении деления на 0 программа сообщает об этом и заканчивает работу.

Процесс записи в иерархический список представлен следующим образом. Список инициализируется первым элементом и подается в функцию его заполнения. Далее идет процесс поиска индексов аргументов. И если найти аргумент-знак не представляет особой задачи, то для других аргументов это будет сделать сложнее. Так, если по ходу просмотра нашего основного выражения встречается скобка или цифра, то программа запоминает нужный индекс, и в том случае, когда это была скобка, она продолжает поиск нужной закрывающей скобки, учитывая количество встретившихся открывающих и закрывающих скобок. Теперь, когда найдены все аргументы, продолжается заполнение списка. Аргументы типа «знак» и «цифра» записываются в элемент структуры списка сразу, а для аргумента «выражение» выделяется дополнительная память под ветвление списка и алгоритм продолжается рекурсивно, переходя уже к новому найденному выражению и рассматривая его как основное выражение.

Процесс упрощения выражения так же начинается с верхушки списка. И так как каждый элемент списка содержит выражение со знаком и аргументами, то применить условия сокращенного вывода достаточно просто, достаточно определить тип аргумента, и в зависимости от аргумента-знака и значения аргументы выводить все выражение или только один из аргументов. Весь вывод так же как и запись происходит рекурсивно. Главное, чтобы проход списка осуществлялся слева-направо, так вывод будет корректен.

**Описание функций**

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Выходной параметр** | **Входные данные** | **Описание** |
| **abort\_** | - | **Const char \* –** анализируемая строка | Выводит сообщение об ошибке и завершает программу аварийно. |
| **fill\_list** | - | **Node \* –** анализируемое выражение;  **Char \* –** анализируемая строка | Заполняет иерархический список. |
| **find\_args** | - | **Char \* –** анализируемая строка | Находит аргументы выражения и запоминает их. |
| **find\_sec\_bracket** | int | **Char \* –** анализируемая строка | Ищет нужную закрывающую скобку для найденной открытой скобки. |
| **fill\_node** | - | **Node \* –** анализируемое выражение;  **Int \* –** массив найденных индексов аргументов;  **Char \* –** анализируемая строка | Заполняет структуру элемента списка пользуясь массивом индексов аргументов. |
| **fill\_expr** | - | **Node \* –** анализируемое выражение; | Оперирует выражениями, посылая их в функцию заполнения. |
| **print\_list** | - | **Node \* –** анализируемое выражение; | Печатает новое сокращенное выражение с добавленным множеством условий. |

**Описание структур данных**

|  |  |  |
| --- | --- | --- |
| **Название** | **Поле** | **Описание** |
| Node | int flag\_type[2]; | Принимает значение 0, если в элементе массива соответствующем порядку объединения находится число и 2 – если указатель на структуру. |
| char sign ; | Здесь хранится знак. |
| union {  int one;  struct Node\* left;  } | Либо число, либо указатель на следующий элемент списка. |
| union {  int two;  struct Node\* right;  } | Либо число, либо указатель на следующий элемент списка. |

**Тестирование.**

|  |  |
| --- | --- |
| Исходное выражение: | Результат: |
| (+ (- 0 4) (/ (+ 0 (\* 5 1)) 1)) | (+ 4 5) |
| (\* 1(\* 1 1)) | 1 |
| (+ (- (\* 1 5) 6) (/ 5 1)) | (+ (- 5 6) 5) |
| (- 0 0) | 0 |
|  |  |
| (+ 6(- (\* 1 5) 0)) | Error - deviision by zero! |
| (/ (- 1 2) 0) | (- 1 2) |

**Вывод.**

В процессе выполнения лабораторной работы были продуманы, созданы и реализованы на практике алгоритмы и методы работы по построению иерархического списка, а также по работе с ним. С помощью построения списка в памяти и вывода его на стандартный поток мы смогли проверить на корректность префиксное арифметическое выражение.

**Приложение 1. Код программы.**

* [**Run\_test.sh**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)

#!/bin/bash

cd ./Sourse

make

cd ..

echo -e '\_\_\_\_\_\_\_\nTest 1:'

cat ./Tests/Test1.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Sourse/Lab2 < ./Tests/Test1.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 2:'

cat ./Tests/Test2.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Sourse/Lab2 < ./Tests/Test2.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 3:'

cat ./Tests/Test3.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Sourse/Lab2 < ./Tests/Test3.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 4:'

cat ./Tests/Test4.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Sourse/Lab2 < ./Tests/Test4.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 5:'

cat ./Tests/Test5.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Sourse/Lab2 < ./Tests/Test5.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 6:'

cat ./Tests/Test6.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Sourse/Lab2 < ./Tests/Test6.txt

echo -e ''

echo -e '\_\_\_\_\_\_\_\nTest 7:'

cat ./Tests/Test7.txt

echo -e '\_\_\_\_\_\_\_\nTesting:\n'

./Sourse/Lab2 < ./Tests/Test7.txt

* [**API.c**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdarg.h>

#include <ctype.h>

#include "API.h"

void print\_list(Node \*node) { //Printting function list

if (node->flag\_type[1] == 1) //Case of division by zero

if (node->sign == '/' && node->two == 0)

abort\_("\nError - deviision by zero!"); //Call error func with message

if(node->sign == '+' && node->flag\_type[0] == 1) //Case of odd or subtraction left 0

if(node->one == 0) {

if(node->flag\_type[1] == 1)

printf(" %d", node->two);

else

print\_list(node->right);

return;

}

if((node->sign == '+' || node->sign == '-') && node->flag\_type[1] == 1) //Case of odd or subtraction right 0

if(node->two == 0) {

if(node->flag\_type[0] == 1)

printf(" %d", node->one);

else

print\_list(node->left);

return;

}

if(node->sign == '\*' && node->flag\_type[0] == 1) //Case of multiplicate on left 1

if(node->one == 1) {

if(node->flag\_type[1] == 1)

printf(" %d", node->two);

else

print\_list(node->right);

return;

}

if(node->sign == '\*' && node->flag\_type[1] == 1) //Case of multiplicate on right 1

if(node->two == 1) {

if(node->flag\_type[0] == 1)

printf(" %d", node->one);

else

print\_list(node->left);

return;

}

if(node->sign == '/' && node->flag\_type[1] == 1) //Case division by 1

if(node->two == 1) {

if(node->flag\_type[0] == 1)

printf(" %d", node->one);

else

print\_list(node->left);

return;

}

printf(" (%c", node->sign); //Print sign

if (node->flag\_type[0] == 2)

print\_list(node->left); //Print left exprission

else

printf(" %d", node->one); //Print left arg

if(node->flag\_type[1] == 2)

print\_list(node->right); //Print right expression

else

printf(" %d", node->two); //Print right exprission

printf(")");

}

void abort\_(const char \* s, ...) //Consulting error with message

{

va\_list args;

va\_start(args, s);

vfprintf(stderr, s, args);

fprintf(stderr, "\n");

va\_end(args);

abort();

}

void fill\_list(Node \* node, char \*str) { //Filling function list

int args[5];

find\_args(str, args);

fill\_node(node, args, str);

}

void fill\_node(Node \* node, int \*args, char \*str){ //Filling finction node

node->sign = str[args[0]]; //Set value sign

node->left = (Node \*)malloc(sizeof(Node)); //Allocate memory to left exprission

node->right = (Node \*)malloc(sizeof(Node)); //Allocate memory to right exprission

if (isdigit(str[args[1]])) { //If first arg os digit

node->flag\_type[0] = 1; //Set type first arg 1

node->one = 0; //Set valuse first arg 0

while (isdigit(str[args[1]])) //Ramp up first arg while we meet digit

node->one = node->one \* 10 + str[args[1]++] - '0';

if (isdigit(str[args[2]])) { //If second arg is digit

node->flag\_type[1] = 1; //Set type second arg 1

node->two = 0; //Set value second arg 0

while (isdigit(str[args[2]])) //Ramp up second arg while we meet digit

node->two = node->two \* 10 + str[args[2]++] - '0';

}

else { //If second arg is expression

node->flag\_type[1] = 2; //Set type second arg 2

fill\_expr(node->right, args[2], args[3], str); //Fill next exprission

}

}

else {

node->flag\_type[0] = 2; //Set type first arg 2

fill\_expr(node->left, args[1], args[2], str); //Fill next exprission

if (isdigit(str[args[3]])) { //If second arg is digit

node->flag\_type[1] = 1; //Set type second arg 1

node->two = 0; //Set value second arg 0

while (isdigit(str[args[3]])) //Ramp up second arg while we meet digit

node->two = node->two \* 10 + str[args[3]++] - '0';

}

else { //If second arg is exprission

fill\_expr(node->right, args[3], args[4], str);

node->flag\_type[1] = 2; //Set tyoe second arg 2

}

}

}

void fill\_expr(Node \*node,int arg1, int arg2, char \*str) { //Filling function exprission

int odd = arg2 - arg1 + 1; //Set odd

char \*new\_str = (char \*)malloc(sizeof(char) \* odd);

strncpy(new\_str, str + arg1, odd);

printf("Component expr %s\n", new\_str);

fill\_list(node, new\_str); //Continue fill list

}

void find\_args(char \*str, int \*args) { //Finding function args

for (int i = 0; i < strlen(str); i++) //Find index sign

if (str[i] == '+' || str[i] == '-' || str[i] == '\*' || str[i] == '/') {

args[0] = i;

break;

}

int limit = 3; //Number of args

int add = args[0]; //Deletion ariable from the beginning of string

for (int i = 1; i < limit; i++) {

args[i] = add + 1; //Set next arg to next index

while (str[args[i]] == ' ') { //Skip spaces

args[i]++;

}

add = args[i]; //Set deletion to current arg

if (isdigit(str[args[i]]) && isdigit(str[args[i]+1])) { //While we meet digit ramp up deletion

while (isdigit(str[add]))

add++;

}

if (str[args[i]] == '(') { //If we meet openong bracket

i++;

args[i] = find\_sec\_bracket(str + args[i-1] + 1) + args[i-1]; //Set next arg to index closing bracket

limit++; //Increment number args

add += args[i] - args[i-1] ; //Add deletion

}

}

}

int find\_sec\_bracket(char \*str) { //Finding function second bracket

int left\_br = 1, right\_br = 0;

int i;

for (i = 0; left\_br != right\_br; i++) { //While left bracket doesn't right bracket continue //While the number of left and right brackets does'nt equal

if (str[i] == '(')

left\_br++;

if (str[i] == ')')

right\_br++;

}

return i; //return deleting

}

* [**API.h**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)

typedef struct Node { //Struct node of list

int flag\_type[2]; //Array keeps type of args: 1 - int, 2 - struct Node \*

char sign; //Arg-sign

union { //First arg

int one; //May be digit

struct Node \*left; //May be expression

};

union { //Second arg

int two; //May be digit

struct Node \*right; //May be expression

};

} Node;

typedef struct List { //Struct of list

Node \*top;

} List;

void abort\_(const char \* , ...); //Consulting error with message

void fill\_list(Node \*, char \*); //Filling function list

void find\_args(char \*, int \*); //Finding function args

int find\_sec\_bracket(char \*); //Finding function second bracket

void fill\_node(Node \*, int \*, char \*); //Filling finction node

void fill\_expr(Node \*,int , int ,char \*); //Filling function exprission

void print\_list(Node \*); //Printting function list

* [**Makefile**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)

all:app

rm API.o Lab2.o

app: Lab2.o API.o

gcc Lab2.o API.o -o Lab2

myk.o: Lab2.c API.h

gcc -c Lab2.c

API.o: API.c API.h

gcc -c API.c