|  |  |
| --- | --- |
|  | **Отчёт по лабораторной работе** № 24 по курсу 1  Практикум на ЭВМ  студента группы M80-104б-18 Сыроежкина Кирилла Геннадьевича, № по списку 18  Адреса www, e-mail, jabber, skype KrillsA@yandex.ru  Работа выполнена: “14“ мая 2019г.  Преподаватель: Доцент каф.806 Никулин С.П.  Входной контроль знаний с оценкой  Отчёт сдан “ “ 2019 г., итоговая оценка  Подпись преподавателя |

* **Тема**: Алгоритмы и структуры данных
* **Цель работы**: Составить программу выполнения заданных преобразований арифметических выражений с применением деревьев.
* **Задание** (*16*): Урать из частных все делители равные единице
* **Оборудование** (*лабораторное*):

ЭВМ 1 , процессор Intel Celeron i686 , имя узла сети client 1 с ОП 1000 МБ

НМД 70 ГБ. Терминал lxterminal адрес: 192.168.2.37 . Принтер

Другие устройства

*Оборудование ПЭВМ студента, если использовалось:*

Процессор Intel core i7-7700 , ОП 16384 МБ, НМД 1024 ГБ. Монитор BENQ GW2470

Другие устройства

* **Программное обеспечение** (*лабораторное*):

Операционная система семейства UNIX, наименование Ubuntu версия 16.04

Интерпретатор команд bash версия

Система программирования Си версия

Редактор текстов emacs версия

Утилиты операционной системы cmp ,comm, wc, dd, diff, grep, join, sort ,tail, tee, tr, uniq, od, sum

Прикладные системы и программы gnuplot, bc

Местонахождения и имена файлов программ и данных /std/188237

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства Windows , наименование Windows 10 версия 10.0.17763.316

Интерпретатор команд cmd версия

Система программирования Си версия

Редактор текстов Sublime text 3 версия 3.1.1

Утилиты операционной системы проводник

Прикладные системы и программы Yandex Browser, notepad++

Местонахождения и имена файлов программ и данных C:\Kirill

* **Идея, метод, алгоритм**

Перевести методом Дейкстры из инфиксной записи в постфиксную:

*Пока не все токены обработаны:*

*Прочитать токен.*

*Если токен — число, то добавить его в очередь вывода.*

*Если токен — функция, то поместить его в стек.*

*Если токен — разделитель аргументов функции (например запятая):*

*Пока токен на вершине стека не открывающая скобка, перекладывать операторы из стека в выходную очередь. Если в стеке не было открывающей скобки, то в выражении пропущен разделитель аргументов функции (запятая), либо пропущена открывающая скобка.*

*Если токен — оператор op1, то:*

*Пока присутствует на вершине стека токен оператор op2, и*

*Либо оператор op1 лево-ассоциативен и его приоритет меньше, чем у оператора op2 либо равен,*

*или оператор op1 право-ассоциативен и его приоритет меньше, чем у op2,*

*переложить op2 из стека в выходную очередь;*

*(Иначе, когда стек операторов пуст или содержит открывающую скобку)*

*положить op1 в стек.*

*Если токен — открывающая скобка, то положить его в стек.*

*Если токен — закрывающая скобка:*

*Пока токен на вершине стека не является открывающей скобкой, перекладывать операторы из стека в выходную очередь.*

*Выкинуть открывающую скобку из стека, но не добавлять в очередь вывода.*

*Если токен на вершине стека — функция, добавить её в выходную очередь.*

*Если стек закончился до того, как был встречен токен открывающая скобка, то в выражении пропущена скобка.*

*Если больше не осталось токенов на входе:*

*Пока есть токены операторы в стеке:*

*Если токен оператор на вершине стека — скобка, то в выражении присутствует незакрытая скобка.*

*Переложить оператор из стека в выходную очередь.*

*Конец.*

Далее создать стек узлов и по следующему алгоритму создать дерево:

*если не достигнут конец строки ввода, прочитать очередной символ, если этот символ - операнд, то занести его в стек1), иначе (символ - операция):*

*1) создать новый элемент, записать в него эту операцию;*

*2) достать из стека два верхних (последних) элемента, присоединить их в качестве левого и правого операндов в новый элемент;*

*3) занести полученный "треугольник" в стек.*

*По окончании работы этого алгоритма в стеке будет содержаться ровно один элемент - указатель на корень построенного дерева*

Обходим дерево и удаляем ненужные ноды

* **Сценарий выполнения работы**

Создать функции для стека строк;

Реализовать алгоритм Дейкстры для создания постфиксной записи;

Записать постфиксную запись в строковый стек и отразить его;

Создать функции для стека нод;

Используя вышеупомянутый алгоритм, стек нод и стек строк, создать дерево;

создать функцию удаление единичных делителей;

создать вспомогательные функцию;

Допущен к выполнению работы. Подпись преподавателя

* **Распечатка протокола**

**kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab24$ cat i.c**

**#include <string.h>**

**#include <stdio.h>**

**#include <stdbool.h>**

**#include <stdlib.h>**

**#define is\_operator(c) (c == '+' || c == '-' || c == '/' || c == '\*' || c == '^')**

**#define is\_ident(c) ((c >= '0' && c <= '9') || (c >= 'a' && c <= 'z'))**

**struct stack\_item\_string // функции для стека строк**

**{**

**char data[40];**

**struct stack\_item\_string\* prev;**

**};**

**typedef struct**

**{**

**struct stack\_item\_string\* top;**

**int size;**

**}stack\_string;**

**void create\_stack\_string(stack\_string \*s) // присвоить начальные значения**

**{**

**s->size=0;**

**s->top=0;**

**}**

**bool empty\_stack\_string(stack\_string\* s) // проверка на пустоту**

**{**

**return s->top==0;**

**}**

**int size\_stack\_string(stack\_string\* s) // размер стека**

**{**

**return s->size;**

**}**

**bool push\_in\_stack\_string(stack\_string\* s, char\* data) // занос нового элемента в стек**

**{**

**struct stack\_item\_string\* tmp = malloc(sizeof(struct stack\_item\_string));**

**if(!tmp)**

**return false;**

**strcpy(tmp->data,data);**

**tmp->prev=s->top;**

**s->top=tmp;**

**s->size++;**

**return true;**

**}**

**bool pop\_from\_stack\_string(stack\_string\* s) // изъятие первого элемента из стека**

**{**

**if (!s->size)**

**return false;**

**struct stack\_item\_string\* tmp = s->top;**

**s->top=s->top->prev;**

**s->size--;**

**free(tmp);**

**return true;**

**}**

**char\* top\_stack\_string(stack\_string\* s) // первый элемент**

**{**

**if(s->top)**

**return s->top->data;**

**}**

**void destroy\_stack\_string(stack\_string\* s) // удалить стек**

**{**

**while(s->top)**

**{**

**struct stack\_item\_string\* tmp=s->top;**

**s->top=s->top->prev;**

**free(tmp);**

**}**

**s->top=0;**

**s->size=0;**

**}**

**void reverse\_stack\_string(stack\_string \*s) // отразить стек**

**{**

**stack\_string tmp1;**

**stack\_string tmp2;**

**create\_stack\_string(&tmp1);**

**create\_stack\_string(&tmp2);**

**while (!empty\_stack\_string(s))**

**{**

**push\_in\_stack\_string(&tmp1, top\_stack\_string(s));**

**pop\_from\_stack\_string(s);**

**}**

**while (!empty\_stack\_string(&tmp1))**

**{**

**push\_in\_stack\_string(&tmp2, top\_stack\_string(&tmp1));**

**pop\_from\_stack\_string(&tmp1);**

**}**

**while (!empty\_stack\_string(&tmp2))**

**{**

**push\_in\_stack\_string(s, top\_stack\_string(&tmp2));**

**pop\_from\_stack\_string(&tmp2);**

**}**

**destroy\_stack\_string(&tmp1);**

**destroy\_stack\_string(&tmp2);**

**}**

**int priority(const char c) // Приоритеты операторов**

**{**

**switch(c)**

**{**

**case '^':**

**return 4;**

**case '\*':**

**case '/':**

**return 3;**

**case '+':**

**case '-':**

**return 2;**

**}**

**return 0;**

**}**

**bool op\_left\_assoc(const char c)**

**{**

**switch(c)**

**{**

**// лево-ассоциативные операторы**

**case '\*':**

**case '/':**

**case '+':**

**case '-':**

**return true;**

**// право-ассоциативные операторы**

**case '^':**

**return false;**

**}**

**return false;**

**}**

**bool infix\_to\_postfix\_algorithm(const char \*input, char \*output, int size\_input)**

**{**

**const char \*strpos = input, \*strend = input + strlen(input);**

**char c, stack[size\_input], sc, \*outpos = output;**

**unsigned int sl = 0;**

**while(strpos < strend)**

**{**

**c = \*strpos;**

**if(c != ' ')**

**{**

**// Если токен является числом (идентификатором), то добавить его в очередь вывода.**

**if(is\_ident(c))**

**{**

**\*outpos = c; ++outpos;**

**}**

**// Если токен оператор op1, то:**

**else if(is\_operator(c))**

**{**

**\*outpos=' '; ++outpos;**

**while(sl > 0)**

**{**

**sc = stack[sl - 1];**

**// Пока на вершине стека присутствует токен оператор op2,**

**// а также оператор op1 лево-ассоциативный и его приоритет меньше или такой же чем у оператора op2,**

**// или оператор op1 право-ассоциативный и его приоритет меньше чем у оператора op2**

**if(is\_operator(sc) &&**

**((op\_left\_assoc(c) && (priority(c) <= priority(sc))) ||**

**(!op\_left\_assoc(c) && (priority(c) < priority(sc)))))**

**{**

**// Переложить оператор op2 из стека в очередь вывода.**

**\*outpos=' '; ++outpos;**

**\*outpos = sc; ++outpos;**

**sl--;**

**}**

**else**

**{**

**break;**

**}**

**}**

**// положить в стек оператор op1**

**\*outpos=' '; ++outpos;**

**stack[sl] = c;**

**++sl;**

**}**

**// Если токен - левая круглая скобка, то положить его в стек.**

**else if(c == '(')**

**{**

**stack[sl] = c;**

**++sl;**

**}**

**// Если токен - правая круглая скобка:**

**else if(c == ')')**

**{**

**bool pe = false;**

**// До появления на вершине стека токена "левая круглая скобка"**

**// перекладывать операторы из стека в очередь вывода.**

**while(sl > 0)**

**{**

**sc = stack[sl - 1];**

**if(sc == '(')**

**{**

**pe = true;**

**break;**

**}**

**else**

**{**

**\*outpos=' '; ++outpos;**

**\*outpos = sc; ++outpos;**

**sl--;**

**}**

**}**

**// Если стек кончится до нахождения токена левая круглая скобка, то была пропущена скобка.**

**if(!pe)**

**{**

**printf("Error: parentheses mismatched\n");**

**return false;**

**}**

**// выкидываем токен "левая круглая скобка" из стека (не добавляем в очередь вывода).**

**sl--;**

**// Если на вершине стека токен - функция, положить его в очередь вывода.**

**}**

**else**

**{**

**printf("Unknown token %c\n", c);**

**return false; // Unknown token**

**}**

**}**

**++strpos;**

**}**

**// Когда не осталось токенов на входе:**

**// Если в стеке остались токены:**

**while(sl > 0)**

**{**

**sc = stack[sl - 1];**

**if(sc == '(' || sc == ')')**

**{**

**printf("Error: parentheses mismatched\n");**

**return false;**

**}**

**\*outpos = sc; ++outpos;**

**--sl;**

**}**

**\*outpos = 0; // Добавляем завершающий ноль к строке**

**return true;**

**}**

**char posfix\_stack\_string(stack\_string \*s) //перенос постфиксной записи в стек c конца в начало (<-)**

**{**

**int size\_input=1, i=0;**

**char \*input = malloc(size\_input), c, tmp[40];**

**tmp[0]=0;**

**printf("\n Введие арифметическое выражение: ");**

**while ((c=getchar()) != EOF)**

**{**

**size\_input++;**

**input = realloc(input, size\_input);**

**sprintf(input, "%s%c", input, c);**

**}**

**printf("\n\nВы ввели: %s", input);**

**char \*output=malloc(size\_input\*2);**

**infix\_to\_postfix\_algorithm(input, output, size\_input);**

**while (output[i]!='\0') // прочитываем постфиксную строку**

**{**

**if (is\_ident(output[i])) // если это цифра то добавляем её в временную строковую переменную**

**{**

**sprintf(tmp, "%s%c", tmp, output[i]);**

**}**

**else if (is\_operator(output[i]))**

**{**

**if (atoi(tmp)!=0) // если в строке число то кладем его в стек и очищаем переменную**

**push\_in\_stack\_string(&\*s, tmp);**

**tmp[0]=0;**

**sprintf(tmp, "%s%c", tmp, output[i]); // кладем оператор в строку и кидаем в стек, а затем очищаем**

**push\_in\_stack\_string(&\*s, tmp);**

**tmp[0]=0;**

**}**

**else if (output[i]=' ' && atoi(tmp)!=0) // если разделитель и в строке число кидаем число в стек и очищаем переменную**

**{**

**push\_in\_stack\_string(&\*s, tmp);**

**tmp[0]=0;**

**}**

**i++;**

**}**

**}**

**typedef struct tree //функции для дерева**

**{**

**char key[40];**

**struct tree\* left;**

**struct tree\* right;**

**}node;**

**struct stack\_item\_node // функции для стека из узлов**

**{**

**node\* data;**

**struct stack\_item\_node\* prev;**

**};**

**typedef struct**

**{**

**struct stack\_item\_node\* top;**

**int size;**

**}stack\_node;**

**void create\_stack\_node(stack\_node \*s) // присвоить начальные значения**

**{**

**s->size=0;**

**s->top=NULL;**

**}**

**bool empty\_stack\_node(stack\_node \*s) // проверка на пустоту**

**{**

**return s->top==NULL;**

**}**

**int size\_stack\_node(stack\_node \*s) // размер стека**

**{**

**return s->size;**

**}**

**bool push\_in\_stack\_node(stack\_node\* s, node \*data) // занос нового элемента в стек**

**{**

**struct stack\_item\_node\* tmp = malloc(sizeof(struct stack\_item\_node));**

**if(!tmp)**

**return false;**

**tmp->data=data;**

**tmp->prev=s->top;**

**s->top=tmp;**

**s->size++;**

**return true;**

**}**

**bool pop\_from\_stack\_node(stack\_node\* s) // изъятие первого элемента из стека**

**{**

**if (!s->size)**

**return false;**

**struct stack\_item\_node\* tmp = s->top;**

**s->top=s->top->prev;**

**s->size--;**

**free(tmp);**

**return true;**

**}**

**node\* top\_stack\_node(stack\_node \*s) // первый элемент**

**{**

**if(s->top)**

**return s->top->data;**

**else**

**return NULL;**

**}**

**void destroy\_stack\_node(stack\_node\* s) // удалить стек**

**{**

**while(s->top)**

**{**

**struct stack\_item\_node\* tmp=s->top;**

**s->top=s->top->prev;**

**free(tmp);**

**}**

**s->top=NULL;**

**s->size=0;**

**}**

**node\* string\_to\_node(char \*s) //преобразование строки в ноду с ключом в виде этой строки**

**{**

**node\* tmp=(node\*)malloc(sizeof(node));**

**strcpy(tmp->key, s);**

**tmp->left=NULL;**

**tmp->right=NULL;**

**return tmp;**

**}**

**node\* string\_to\_tree(stack\_string \*input) //преобразование стека строк в дерево**

**{**

**stack\_node tree;**

**node\* tmp=(node\*)malloc(sizeof(node));**

**create\_stack\_node(&tree);**

**while (!empty\_stack\_string(input))**

**{**

**if (atoi(top\_stack\_string(input))!=0)**

**push\_in\_stack\_node(&tree, string\_to\_node(top\_stack\_string(input)));**

**else**

**{**

**tmp=string\_to\_node(top\_stack\_string(input));**

**tmp->right=top\_stack\_node(&tree);**

**pop\_from\_stack\_node(&tree);**

**tmp->left=top\_stack\_node(&tree);**

**pop\_from\_stack\_node(&tree);**

**push\_in\_stack\_node(&tree, tmp);**

**}**

**pop\_from\_stack\_string(input);**

**}**

**tmp=top\_stack\_node(&tree);**

**destroy\_stack\_node(&tree);**

**return tmp;**

**}**

**void tree\_print(node\* tree, int lrc) //печать дерева**

**{**

**static int level = 0;**

**int i;**

**level++;**

**if (tree){**

**tree\_print(tree->right, 2);**

**for (i = 0; i<level; i++) printf(" ");**

**if(lrc == 1) printf("\\\_%s\n", tree->key);**

**else if (lrc == 2) printf("\_\_%s\n", tree->key);**

**else printf("\_%s\n", tree->key);**

**tree\_print(tree->left, 1);**

**}**

**level--;**

**}**

**void remove\_div\_one(node\* tree) //удаление в дереве деление на 1**

**{**

**if (tree)**

**{**

**remove\_div\_one(tree->left);**

**remove\_div\_one(tree->right);**

**if (strcmp(tree->key, "/")==0 && strcmp(tree->right->key, "1")==0)**

**{**

**free(tree->right);**

**\*tree=\*tree->left;**

**}**

**}**

**}**

**int pr\_int(char c) //порядок действий**

**{**

**switch(c)**

**{**

**case '-': case '+': return 1;**

**case '\*': case '/': case '^': return 2;**

**}**

**}**

**void print\_infix(node\* tree, int priority\_node) //распечатка инфиксной записи из дерева**

**{**

**if(priority\_node == 2)**

**printf("(");**

**if(tree->left)**

**{**

**if(!tree->left->left && !tree->left->right)**

**print\_infix(tree->left, 1);**

**else**

**print\_infix(tree->left, pr\_int(tree->key[0]));**

**}**

**printf("%s", tree->key);**

**if(tree->right)**

**{**

**if(!tree->right->left && !tree->right->right)**

**print\_infix(tree->right, 1);**

**else**

**print\_infix(tree->right, pr\_int(tree->key[0]));**

**}**

**if(priority\_node == 2)**

**printf(")");**

**}**

**int main()**

**{**

**stack\_string input;**

**node\* tree = (node\*)malloc(sizeof(node));**

**create\_stack\_string(&input);**

**posfix\_stack\_string(&input);**

**reverse\_stack\_string(&input);**

**tree=string\_to\_tree(&input);**

**destroy\_stack\_string(&input);**

**printf("\nПредставление в дереве:\n");**

**tree\_print(tree, 0);**

**remove\_div\_one(tree);**

**printf("\nДерево после преобразований: \n");**

**tree\_print(tree, 0);**

**printf("\nВыражение после преобразований: \n");**

**print\_infix(tree, 1);**

**printf("\n");**

**}**

kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab24$ gcc i.c

kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab24$ ./a.out

Введие арифметическое выражение: 1+2/1

Вы ввели: 1+2/1

Представление в дереве:

\_\_1

\_\_/

\\_2

\_+

\\_1

Дерево после преобразований:

\_\_2

\_+

\\_1

Выражение после преобразований:

1+2

kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab24$ ./a.out

Введие арифметическое выражение: (1+2)/(5-1/1)/1+34\*(3^(6/1))

Вы ввели: (1+2)/(5-1/1)/1+34\*(3^(6/1))

Представление в дереве:

\_\_1

\_\_/

\\_6

\_\_^

\\_3

\_\_\*

\\_34

\_+

\_\_1

\\_/

\_\_1

\_\_/

\\_1

\_\_-

\\_5

\\_/

\_\_2

\\_+

\\_1

Дерево после преобразований:

\_\_6

\_\_^

\\_3

\_\_\*

\\_34

\_+

\_\_1

\_\_-

\\_5

\\_/

\_\_2

\\_+

\\_1

Выражение после преобразований:

(1+2)/(5-1)+34\*(3^6)

kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab24$ ./a.out

Введие арифметическое выражение: (1+2)/1+5\*(7-23/1)/(4\*5)-31^(2+3)-1

Вы ввели: (1+2)/1+5\*(7-23/1)/(4\*5)-31^(2+3)-1

Представление в дереве:

\_\_1

\_-

\_\_3

\_\_+

\\_2

\_\_^

\\_31

\\_-

\_\_5

\_\_\*

\\_4

\_\_/

\_\_1

\_\_/

\\_23

\_\_-

\\_7

\\_\*

\\_5

\\_+

\_\_1

\\_/

\_\_2

\\_+

\\_1

Дерево после преобразований:

\_\_1

\_-

\_\_3

\_\_+

\\_2

\_\_^

\\_31

\\_-

\_\_5

\_\_\*

\\_4

\_\_/

\_\_23

\_\_-

\\_7

\\_\*

\\_5

\\_+

\_\_2

\\_+

\\_1

Выражение после преобразований:

1+2+(5\*(7-23))/(4\*5)-31^(2+3)-1

Вы ввели: 99986+1231234/1\*23123-8912894^124124-123+((23-1))

Представление в дереве:

\_\_1

\_\_-

\\_23

\_+

\_\_123

\\_-

\_\_124124

\_\_^

\\_8912894

\\_-

\_\_23123

\_\_\*

\_\_1

\\_/

\\_1231234

\\_+

\\_99986

Дерево после преобразований:

\_\_1

\_\_-

\\_23

\_+

\_\_123

\\_-

\_\_124124

\_\_^

\\_8912894

\\_-

\_\_23123

\_\_\*

\\_1231234

\\_+

\\_99986

Выражение после преобразований:

99986+1231234\*23123-8912894^124124-123+23-1

kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab24$ ./a.out

Введие арифметическое выражение: 1/3+4

Вы ввели: 1/3+4

Представление в дереве:

\_\_4

\_+

\_\_3

\\_/

\\_1

Дерево после преобразований:

\_\_4

\_+

\_\_3

\\_/

\\_1

Выражение после преобразований:

1/3+4

Введие арифметическое выражение: 1/1/1/1/1/1/1/1

Вы ввели: 1/1/1/1/1/1/1/1

Представление в дереве:

\_\_1

\_/

\_\_1

\\_/

\_\_1

\\_/

\_\_1

\\_/

\_\_1

\\_/

\_\_1

\\_/

\_\_1

\\_/

\\_1

Дерево после преобразований:

\_1

Выражение после преобразований:

1

kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab24$ ./a.out

Введие арифметическое выражение: 823\*(8^(9\*9))

Вы ввели: 823\*(8^(9\*9))

Представление в дереве:

\_\_9

\_\_\*

\\_9

\_\_^

\\_8

\_\*

\\_823

Дерево после преобразований:

\_\_9

\_\_\*

\\_9

\_\_^

\\_8

\_\*

\\_823

Выражение после преобразований:

823\*(8^(9\*9))

kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab24$ ./a.out

Введие арифметическое выражение: 2/1-2

Вы ввели: 2/1-2

Представление в дереве:

\_\_2

\_-

\_\_1

\\_/

\\_2

Дерево после преобразований:

\_\_2

\_-

\\_2

Выражение после преобразований:

2-2

* **Дневник отладки**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб.  или  дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
| 1 | дом | 10.5.13 | 19:00 | Не во всех случаях работал алгоритм Дейкстры | Внимательно ещё раз просмотрел работу алгорима. Проблема была устранена |  |

* Замечание автора по существу работы теперь я хорошо могу использовать деревья в си
* Выводы Я составил программу выполнения заданных преобразований арифметических выражений с применением деревьев.

устранены следующим образом Внимательное прочтение алгоритмов.

Подпись студента