my\_tree.h

#ifndef \_my\_tree\_h\_

#define \_my\_tree\_h\_

#include "my\_stack.h"

#include "functs.h"

int cnt,amount,lvls;

struct tree{

struct elem item;

struct tree\* right;

struct tree\* left;

int bracket;

};

struct tree\* remove\_tree(struct tree\* root);

struct tree\* build\_tree(struct elem\* ar, int br);

void print\_tree(struct tree\* root, int i);

void print\_from\_tree(struct tree\* root);

void count\_br(struct tree\* root);

#endif

my\_tree.c

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

#include "my\_stack.h"

#include "functs.h"

#include "my\_tree.h"

struct tree\* remove\_tree(struct tree\* root) {

if (root) {

remove\_tree(root->right);

remove\_tree(root->left);

free(root);

}

return NULL;

}

struct tree\* build\_tree(struct elem\* ar, int br) {

struct tree\* res=malloc(sizeof(struct tree));

if(cnt!=-1) {

if(ar[cnt].type==0) {

res->item.type=ar[cnt].type;

res->item.value=ar[cnt].value;

res->item.symb=ar[cnt].symb;

res->bracket=br;

cnt--;

res->right=build\_tree(ar,2);

res->left=build\_tree(ar,1);

}else if (ar[cnt].type==1||ar[cnt].type==2){

res->item.type=ar[cnt].type;

res->item.value=ar[cnt].value;

res->item.symb=ar[cnt].symb;

res->bracket=br;

cnt--;

}

}

return res;

}

void print\_tree(struct tree\* root, int i) {

if (root) {

print\_tree(root->right, i+1);

for(int k=0; k<=i; k++) printf(" ");

if ((root->item.type==0)||(root->item.type==2)) printf("%c\n",root->item.symb);

else printf("%d\n", root->item.value);

print\_tree(root->left, i+1);

}

}

void print\_from\_tree(struct tree\* root) {

if(root->left){

if(root->item.type==0) printf("(");

print\_from\_tree(root->left);

}

if(root->item.type==1) printf("%d", root->item.value);

if(root->item.type==2 || root->item.type==0) printf("%c", root->item.symb);

if(root->right){

print\_from\_tree(root->right);

if(root->item.type==0) printf(")"); }

}

void count\_br(struct tree\* root) {

if(root->left) count\_br(root->left);

if(root->item.type==1) {

if(root->bracket==1) lvls++;

else if(root->bracket==2) lvls--; }

if(root->item.type==2) {

if(root->bracket==1) lvls++;

else if(root->bracket==2) lvls--; }

if(root->right) count\_br(root->right);

}

my\_stack.h

#ifndef \_my\_stack\_h\_

#define \_my\_stack\_h\_

struct node\_s {

char value;

struct node\_s\* prev;

};

typedef struct node\_s Node\_s;

typedef struct node\_s\* Stack;

int empty\_s(Stack s);

char first\_s(Stack s);

int count\_s(Stack s);

Stack in\_s(Stack s, char c);

Stack out\_s(Stack s, char\* c);

Stack del\_s(Stack s);

#endif

my\_stack.c

#include<stdio.h>

#include <stdlib.h>

#include <string.h>

#include<ctype.h>

#include "my\_stack.h"

int empty\_s(Stack s){

if (s==NULL) return 0;

else return 1;

}

char first\_s(Stack s){

if(s!=NULL) return s->value;

else return 0;

}

int count\_s(Stack s){

int i;

for(int i=0;s!=NULL;i++) s=s->prev;

return i;

}

Stack in\_s(Stack s, char c){

Stack ss;

ss=malloc(1\*sizeof(Node\_s));

ss->value=c;

ss->prev=s;

return ss;

}

Stack out\_s(Stack s, char\* c){

\*c=s->value;

Stack ss=s->prev;

free(s);

return ss;

}

Stack del\_s(Stack s){

Stack ss;

while(ss!=NULL) {

ss=s->prev;

free(s);

s=ss;

}

return s;

}

functs.h

#ifndef \_functs\_h\_

#define \_functs\_h\_

#include "my\_stack.h"

int count, check, tree\_size,st;

struct elem{

int type;

int value;

char symb;

};

struct elem\* Dejkstra(struct elem\* part, struct elem\* res);

int size\_of\_str(char\* str);

int check\_expr(char\* str);

struct elem\* parse(char\* ch);

void print\_pars(struct elem\* orig);

#endif

functs.c

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

#include "my\_stack.h"

#include "functs.h"

#define N 100

int is\_var(char k) {

if((k>=65&&k<=90) || (k>=97&&k<=122)) return 1;

else return 0;

}

struct elem\* Dejkstra(struct elem\* part, struct elem\* res){

int i=0,j=0,state=1;

char m,c,t;

Stack temp=NULL;

for (i=0;i<count;i++) {

if(part[i].type==1||part[i].type==2) { res[j]=part[i]; ++j; }

if (part[i].type==3) temp=in\_s(temp,part[i].symb);

if (part[i].type==4) {

while((c=first\_s(temp))!='(') {

temp=out\_s(temp,&m);

res[j].symb=m;

++j;

}

temp=out\_s(temp,&m);

}

if(part[i].type==0){

t=part[i].symb;

if(temp!=NULL){

c=first\_s(temp);

while(((t=='+'||t=='-')&&(c=='+'||c=='-'||c=='\*'||c=='/'||c=='^'))||((t=='\*')&&(c=='\*'||c=='/'||c=='^'))||((t=='/')&&(c=='^'||c=='\*'||c=='/'))){

temp=out\_s(temp,&m);

res[j].symb=m;

++j;

c=first\_s(temp);

}

temp=in\_s(temp,t);

}else temp=in\_s(temp,t); }

}

while(empty\_s(temp)) {

temp=out\_s(temp,&m);

res[j].symb=m;

++j;

}

count=j;

return res;

}

int size\_of\_str(char\* str) {

int i=0;

while(str[i]!='\0') i++;

return i;

}

int check\_expr(char\* str) {

int i=0,s=0;

if(str[0]=='/'||str[0]=='\*'||str[0]=='^') return 0;

count = size\_of\_str(str);

for(i=0;i<count;i++){

if( str[i]<=39 || str[i]==44 || str[i]==46 || (str[i]>=58&&str[i]<=64) || (str[i]>=91&&str[i]<=93) || str[i]==95 || str[i]==96 || (str[i]>=123&&str[i]<=127) )return 0;

if(str[i]=='(') s++;

if(str[i]==')') s--;

if(((str[i]>=48&&str[i]<=57)||is\_var(str[i]))&&(str[i+1]=='(')) return 0;

if((str[i]==')')&&((str[i+1]>=48&&str[i+1]<=57)||(str[i+1]>=65&&str[i+1]<=90)||(str[i+1]>=97&&str[i+1]<=122))) return 0;

if((str[i]=='(')&&(str[i+1]=='\*'||str[i+1]=='/'||str[i+1]=='^'||str[i+1]==')')) return 0;

if((str[i]=='+'||str[i]=='-'||str[i]=='\*'||str[i]=='/'||str[i]=='^')&&(str[i+1]==')')) return 0;

if((str[i]=='+'||str[i]=='-'||str[i]=='\*'||str[i]=='/'||str[i]=='^')&&(str[i-1]=='+'||str[i-1]=='-'||str[i-1]=='\*'||str[i-1]=='/'||str[i-1]=='^'))return 0;

}

if(s!=0)return 0;

return 1;

}

struct elem\* parse(char\* ch) {

struct elem \*part;

part=malloc(N\*sizeof(struct elem));

int i=0,j=0,k=0,b=-1,a=-1,l=0;

char temp[N];

for(int i=0;i<N;i++)temp[i]=0;

i=0;

if((ch[0]=='-'||ch[0]=='+')&&(isdigit(ch[1]))){

i++;

part[j].type=1;

k=0;

while((isdigit(ch[i]))){

temp[k]=ch[i];

k++;

i++;

}

b=atoi(temp);

if(ch[0]=='-') b=b\*(-1);

part[j].value=b;

++j;

for(a=0;a<N;a++)temp[a]='\0';

}

if( (ch[0]=='-'||ch[0]=='+') && ( is\_var(ch[1]) || ch[1]=='(' ) ){

if(is\_var(ch[1])){ part[j].type=3; part[j].symb='('; j++; }

part[j].type=1;

part[j].value=0;

j++;

part[j].type=0;

if(ch[0]=='-') part[j].symb='-';

else part[j].symb='+';

j++;

i++;

if(is\_var(ch[1])){

part[j].type=2;

part[j].symb=ch[i];

j++;

i++;

part[j].type=4;

part[j].symb=')';

j++;

}

}

while(ch[i]!='\0'){

if (ch[i]=='(') {

part[j].type=3;

part[j].symb='(';

++j;

++i;

if((ch[i]=='-'||ch[i]=='+') && ( is\_var(ch[i+1]) || ch[i+1]=='(' )) {

part[j].type=1;

part[j].value=0;

j++;

part[j].type=0;

if(ch[i]=='-') part[j].symb='-';

else part[j].symb='+';

j++;

i++;

}

}

if (ch[i]==')'){ part[j].type=4; part[j].symb=')'; ++j; ++i; }

if (ch[i]=='+'){

if(ch[i-1]=='('){

++i;

part[j].type=1;

k=0;

while((isdigit(ch[i]))){

temp[k]=ch[i];

k++;

i++;

}

b=atoi(temp);

part[j].value=b;

++j;

for(a=0;a<N;a++)temp[a]='\0';

}else{ part[j].type=0; part[j].symb='+'; ++j; ++i; }

}

if (ch[i]=='-'){

if(ch[i-1]=='('){

++i;

part[j].type=1;

k=0;

while((isdigit(ch[i]))){

temp[k]=ch[i];

k++;

i++;

}

b=atoi(temp);

b=b\*(-1);

part[j].value=b;

++j;

for(a=0;a<N;a++)temp[a]='\0';

}else{ part[j].type=0; part[j].symb='-'; ++j; ++i; }

}

if (ch[i]=='\*'){ part[j].type=0; part[j].symb='\*'; ++j; ++i; }

if (ch[i]=='/'){ part[j].type=0; part[j].symb='/'; ++j; ++i; }

if (ch[i]=='^'){ part[j].type=0; part[j].symb='^'; ++j; ++i; }

if((isdigit(ch[i]))){

part[j].type=1;

k=0;

while((isdigit(ch[i]))){

temp[k]=ch[i];

k++;

i++;

}

b = atoi(temp);

part[j].value=b;

++j;

for(a=0;a<N;a++)temp[a]='\0';

}

if(ch[i]>='A'&& ch[i]<='Z' || ch[i]>='a'&& ch[i]<='z'){ part[j].type=2; part[j].symb=ch[i]; ++j; ++i; }

}

count=j;

return part;

}

void print\_pars(struct elem\* orig) {

for(int j=0;j<count;j++){

if (orig[j].type!=1) printf("%c ",orig[j].symb);

else printf("%d ",orig[j].value);

}

printf("\n");

}

24.c

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

#include "my\_stack.h"

#include "functs.h"

#include "my\_tree.h"

#define N 100

int count\_op(struct tree\* root) {

if(root) {

if(root->item.type==0) amount++;

count\_op(root->right);

count\_op(root->left);

}

return amount;

}

int main() {

cnt=0,amount=0, lvls=0;

int i=0, k, b, j=0,a=0, op=0;

char ch[N],res[N];

for(i=0;i<N;i++) ch[i]=0;

printf("Enter expression:\n");

scanf("%s",ch);

if(check\_expr(ch))

{

struct elem \*orig;

struct elem \*ress;

orig=malloc(N\*sizeof(struct elem));

ress=malloc(N\*sizeof(struct elem));

orig=parse(ch);

puts("Print parsed expression:");

print\_pars(orig);

ress=Dejkstra(orig,ress);

puts("Print postfix:");

print\_pars(ress);

struct tree\* T=NULL;

T=malloc(sizeof(struct tree));

T->right=NULL;

T->left=NULL;

T->right=malloc(sizeof(struct tree));

T->left=malloc(sizeof(struct tree));

cnt=count-1;

struct tree\* tr=NULL;

tr=malloc(sizeof(struct tree));

tr->right=NULL;

tr->left=NULL;

tr=build\_tree(ress,0);

puts("Print tree:");

print\_tree(tr,0);

puts("Print expression from tree");

count\_br(tr);

print\_from\_tree(tr);

op=count\_op(tr);

printf("\n\*\*\* Amount of operations - %d \*\*\*\n", op);

} else printf ("Wrong expression\n");

return 0;

}