//stack\_utils.h

//stack\_utils.h

#include <stdio.h>

#include <stdlib.h>

#define MAX (100)

#define TRUE (1)

#define FALSE (0)

#define SUCCESS (1)

#define FAILED (0)

typedef struct stack

{

char item[MAX];

int top;

}stack;

int isEmpty(stack\*);

int isFull(stack\*);

int push(stack\*, char);

char pop(stack\*);

void display(stack\*);

stack\* new\_stack();

int isEmpty(stack \*s)

{

if(s->top == -1) return TRUE;

return FALSE;

}

int isFull(stack \*s)

{

if(s->top == MAX - 1) return TRUE;

return FALSE;

}

int push(stack \*s, char elem)

{

if(isFull(s))

return FAILED;

s->item[++s->top] = elem; return SUCCESS;

}

char pop(stack \*s)

{

if(isEmpty(s)) return FAILED;

return(s->item[s->top--]);

}

void display(stack \*s)

{

if(isEmpty(s)) return;

int i;

for(i = 0; i <= s->top; i++)

printf("%c ", s->item[i]);

printf("\n");

}

stack\* new\_stack()

{

stack\* s = (stack \*)malloc(sizeof(stack)); s->top = -1;

return s;

}

1)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "stack\_utils.h"

int eval(char\*);

int eval(char\* exp)

{

int i;

stack\* s = new\_stack();

for(i = strlen(exp) - 1; i > -1; i--)

{

if(exp[i] >= '0' && exp[i] <= '9')

{

push(s, exp[i]);

}

else if(exp[i] == '+' || exp[i] == '-' || exp[i] == '\*' || exp[i] == '/' || exp[i] == '%')

{

int op1, op2;

op1 = pop(s) - '0';

op2 = pop(s) - '0';

int res; if(exp[i] == '+') res = op1 + op2;

else if(exp[i] == '-')

res = op1 - op2;

else if(exp[i] == '\*')

res = op1 \* op2;

else if(exp[i] == '/')

res = op1 / op2;

else if(exp[i] == '%')

res = op1 % op2;

push(s, res + '0');

}

}

int res = pop(s) - '0';

free(s);

return(res);

}

int main(int argc, char const \*argv[])

{

char\* exp = (char \*)calloc(20, sizeof(char));

printf("This method WILL only work on operands between 0-9\n");

printf("Enter an expression (in prefix): ");

scanf("%s", exp);

printf("Evaluated: %d\n", eval(exp)); return 0;

}

2)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "stack\_utils.h"

void Inf2Pref(char\* , char\*);

void Inf2Post(char\*, char\*);

int isOperand(char ch)

{

return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

int precedence(char ch)

{

switch (ch)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

void Inf2Pref(char\* src, char\* dest)

{

char\* inf\_rev = (char \*)calloc(MAX, sizeof(char));

char\* post\_rev = (char \*)calloc(MAX, sizeof(char));

int i;

int j = 0;

for(i = strlen(src) - 1; i > -1; i--)

{

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

inf\_rev[j] = ')';

else if(src[i] == ')')

inf\_rev[j] = '(';

else

inf\_rev[j] = src[i];

j++;

}

inf\_rev[j] = '\0';

char\* post = (char \*)calloc(MAX, sizeof(char));

Inf2Post(inf\_rev, post);

j = 0;

for(i = strlen(post) - 1; i > -1; i--)

{

post\_rev[j] = post[i];

j++;

}

post\_rev[j] = '\0';

strcpy(dest, post\_rev);

}

void Inf2Post(char\* src, char\* dest)

{

char\* exp = (char \*)calloc(MAX, sizeof(char));

stack\* s = new\_stack();

int i;

int j = 0;

for(i = 0; i < strlen(src); i++)

{

if(isOperand(src[i]))

{

exp[j] = src[i];

j++;

}

else if(src[i] == '(')

{

push(s, src[i]);

}

else if(src[i] == ')')

{

while(isEmpty(s) == FALSE && s->item[s->top] != '(')

{

exp[j] = pop(s);

j++;

}

pop(s);

}

else

{

while(isEmpty(s) == FALSE && precedence(src[i]) <= precedence(s->item[s->top]))

{

exp[j] = pop(s);

j++;

}

push(s, src[i]);

}

}

while(isEmpty(s) == FALSE)

{

exp[j] = pop(s); j++;

}

exp[j] == '\0'; // printf("%s", exp);

strcpy(dest, exp);

}

int main(int argc, char const \*argv[])

{

char\* exp = (char \*)calloc(MAX, sizeof(char));

char\* pref = (char \*)calloc(MAX, sizeof(char));

printf("This WILL only work on operands between A-Z\n");

printf("Enter an expression (in infix): ");

scanf("%s", exp);

Inf2Pref(exp, pref);

printf("The prefix expression is: %s\n", pref); return 0;

}

3)

#include <stdio.h>

#include <stdlib.h>

typedef struct TwoStacks

{

int\* arr; int size; int top1, top2;

}TwoStacks;

TwoStacks\* new\_twoStack(int sz)

{

TwoStacks\* ts = (TwoStacks \*)malloc(sizeof(TwoStacks));

ts->arr = (int \*)calloc(sz, sizeof(int));

ts->top1 = -1;

ts->top2 = sz;

return ts;

}

void push\_stack1(TwoStacks\* ts, int elem)

{

if(ts->top1 < ts->top2 -1)

{

ts->top1++;

ts->arr[ts->top1] = elem;

}

else

{

printf("Overflow\n");

exit(0);

}

}

void push\_stack2(TwoStacks\* ts, int elem)

{

if(ts->top1 < ts->top2 -1)

{

ts->top2--;

ts->arr[ts->top2] = elem;

}

else

{

printf("Overflow\n");

exit(1);

}

}

int pop\_stack1(TwoStacks\* ts, int sz)

{

if(ts->top1 >= 0)

{

int x = ts->arr[ts->top1];

ts->top1--;

return x;

}

else

{

printf("Underflow\n");

exit(1);

}

}

int pop\_stack2(TwoStacks\* ts, int sz)

{

if(ts->top2 < sz)

{

int x = ts->arr[ts->top2];

ts->top2++;

return x;

}

else

{

printf("Underflow\n");

exit(1);

}

}

int main(int argc, char const \*argv[])

{

TwoStacks\* ts = new\_twoStack(5);

push\_stack1(ts, 5);

push\_stack2(ts, 10);

push\_stack2(ts, 15);

push\_stack1(ts, 11);

push\_stack2(ts, 7);

printf("Popped Element from stack 1 is: %d\n", pop\_stack1(ts, 5));

push\_stack2(ts, 40);

printf("Popped Element from stack 2 is %d\n", pop\_stack2(ts, 5));

return 0;

}